ACSL AS A PARALLEL SIMULATION LANGUAGE

SPECIAL TECHNICAL REPORT

REPORT NO. STR-0142-90-007

January 1990

GUIDANCE, NAVIGATION AND CONTROL

DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Contract Data Requirements List Item A004

Period Covered: Not Applicable

Type Report: As Required

ADA

[redacted]

DISCLAIMER

DISCLAIMER STATEMENT - The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

(1) DISTRIBUTION STATEMENT - Approved for public release; distribution is unlimited.

(2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227 - 7013 October 1988.

ACSL AS A PARALLEL SIMULATION LANGUAGE

January 1990

Thomas R. Collins

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332-0540

Eugene L. Sanders
USASDC
Contract Monitor

Cecil O. Alford
Georgia Tech
Project Director

1. BASIC FEATURES OF ACSL

By its very design, ACSL is intended for computer simulation of continuous or continuous/discrete systems. Numerical integration methods are built-in, and provision is made for all of the interacting physical processes that normally take place in parallel between integration steps. A well-written ACSL program is inherently modular, making subsequent use by other programmers much easier. ACSL also has provision for parametric studies, which are often required in simulations, as well as support for various plots, including time response, gain, phase, Nichols, and Nyquist plots.

2. PARALLEL PROGRAMMING IN ACSL

It is possible to write simulations in ACSL in such a way that they can be easily ported to the Parallel Function Processor (PFP), virtually guaranteeing identical output. The procedure to be followed is described below. In general, it consists of using the ACSL PROCEDURAL statement to define the parallelism, and it requires no knowledge of the PFP hardware or software on the part of the ACSL programmer. The PFP programmers will simply verify the ACSL program on their own workstation, then partition the Fortran code generated by the ACSL translator. This partitioning is semi-automated at present and can probably be made fully-automated.

The focus of the parallel programming process is the DERIVATIVE subsection of the DYNAMIC section in the ACSL program. The ACSL programmer begins by identifying sections of code which may be performed in parallel in the ACSL code by surrounding them with a PROCEDURAL statement and a matching END statement, even if some of these sections only include one line. Normally, these sections correspond to a functional unit, such as the IMU or a propulsion system. In the PROCEDURAL statement, all inputs and outputs must be listed. The ACSL compiler does not check these inputs and outputs for correctness in terms of the actual statements within the block. Sometimes ACSL programmers use this to their advantage to force the compiler to sort statements in a certain order, but this cannot be done if the program is eventually to be run in parallel. This is because we cannot allow the sort order to matter (outside of PROCEDURAL blocks, which remain unsorted internally).

Any statements which perform integration should not be placed inside PROCEDURAL blocks. These include the INTEG, INTVC, and LIMINT statements. The integration statements may be grouped together at the end of the program or each may be placed immediately after the PROCEDURAL block which calculates the derivative of the variable being integrated.

The result of this process should be a program in which all statements (except integrations) in the DERIVATIVE section belong to a PROCEDURAL block. One and only one PROCEDURAL block will have any given variable as an output -- the translator will enforce this. This allows the

translator to sort the PROCEDURAL blocks in absolutely any order, while retaining the exact ordering of statements within each block. Since the simulation will run correctly with any ordering, it will also run correctly in parallel. The inputs and outputs given in the PROCEDURAL statements correspond to variables which are received or sent by that process over the crossbar interconnection network, so once again the importance of accuracy in these input/output lists becomes clear: a process will not have access to the required variables if they are not listed.

Each PROCEDURAL block is equivalent to an ADA task, and the input/output list specifies the required communications between tasks. Consequently, the PROCEDURAL ACSL implementation can be viewed as a step in the migration to ADA.

A more subtle aspect of the PROCEDURAL definitions is that, preferably, each PROCEDURAL block should output only derivative variables (i.e., variables which occur as the derivative in some integration statement). This allows each block to run in parallel during the two primary phases of each timestep: 1) derivative evaluation, and 2) integration. This is not a rigid requirement, and it can be worked around during the porting process.

3. CONVERTING FORTRAN PROGRAMS

Some programs, like EXOSIM, have already been written in FORTRAN, and some effort will be required to convert them to ACSL. This is not too difficult for several reasons. First, all FORTRAN subroutines are usable in an ACSL model, probably with no changes. Second, it is actually possible to eliminate some FORTRAN code, since integration is built into ACSL (and corresponding routines exist for the PFP). Finally, if the FORTRAN program is inherently modular, it should translate directly into the PROCEDURAL sections described above.

4. AN EXAMPLE PROGRAM

The example program to be presented here is modified in several stages to illustrate the major points. The result of each step is given as a listing of the ACSL model and is included in Appendices A through E. The ACSL program "missil.csl" is taken directly from the examples given in the ACSL manual. It implements a simple 6-DOF missile with only the basic functional elements. It has no target model, seeker, guidance law, or autopilot, but it is sufficient to illustrate the method.

A block diagram of the model is given as Figure 1. The dotted lines indicate the four partitions which are identified in the following section.

4.1 DEFINING THE MAIN BLOCKS

The statements of missil.csl were rearranged, and PROCEDURAL statements were added to form four main blocks:

Block 0: motor, aerodynamics, and rotational velocity dynamics

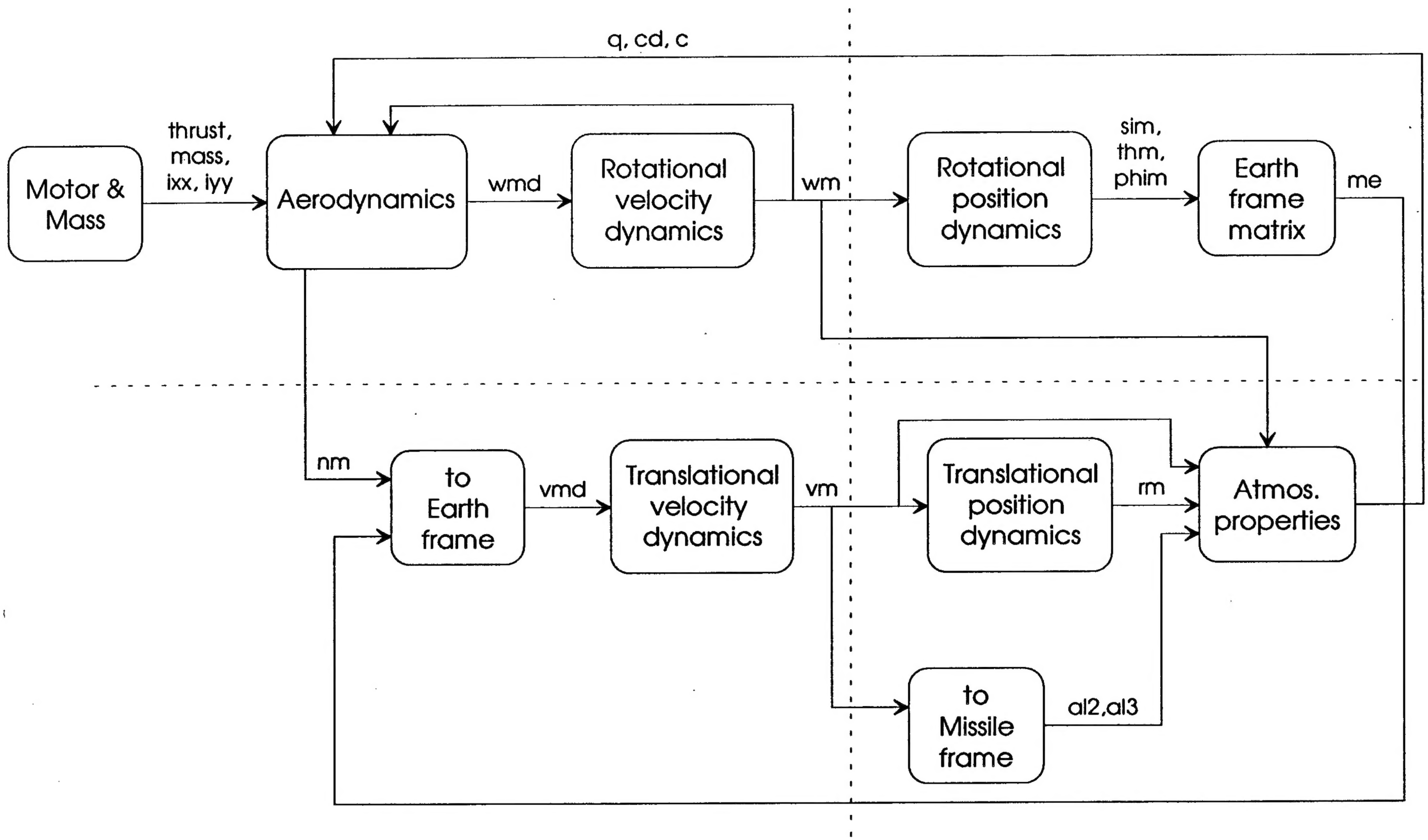

Figure 1: Block diagram of simple missile model
(reference: ACSL Reference Manual, Mitchell and Gauthier Associates)

Block 1: rotational position dynamics

Block 2: translational velocity dynamics

Block 3: translational position dynamics

This particular partitioning is often effective for 6-DOF models. While the individual degrees of freedom may be further split out into smaller parallel blocks, it is often not advantageous, since there is much dependency on certain intermediate variables such as coordinate transformation matrices and aerodynamic coefficients.

The result of this partitioning is missil2.csl, also in the appendix. See the comments in the header and note that all changes to the original program are made in lower case. It is clear that very few changes were made, just some minor rearrangements (and deletions of the original PROCEDURALs, whose orderings were retained within the new PROCEDURALs).

There was no expectation that missil2.csl would run, however, since a circular definition exists. Specifically, by looking just at the PROCEDURAL statements (which is all that ACSL does to determine sort order for this model), it is obvious that NM depends on C, and C in turn depends on NM. This can be easily rectified by taking the statements which define NM(1), NM(2), and NM(3) out of block 0 and putting them in block 2, where they more logically belong. This is shown in the next stage, missil3.csl. This was not done earlier simply because it involved separating statements that were together in one of the example's original PROCEDURAL blocks. A quick inspection of the original program shows that there is no reason why these statements need to precede the statements which define WMD(1), WMD(2), and WMD(3).

Unlike the intermediate step (missil2.csl), missil3.csl will translate, compile, and run, giving the same results as the original model and taking the same time to execute. The difference is that the FORTRAN code generated by ACSL can be ported to four processors on the PFP.

4.2 EMPHASIZING STATE VARIABLES

While this four-processor implementation is usable, it would involve some additional manipulation, probably manually, in order to set up the appropriate communication channels. The problem is that missil3.csl does not adhere to the model of generating only derivative variables as outputs of each PROCEDURAL block. (A corollary to this is that all such blocks must depend only on state variables, or perhaps on other derivative variables).

The next stage, shown in missil4.csl, shows that it is possible to make this missile model adhere to the desired form. This usually involves replicating some code sections that generate convenient intermediate non-state variables that are used in more than one block (in this case, C, CD, and Q). This probably adds little or nothing to the execution time of the parallel implementation on the PFP in this case, since the replicated lines are added to block 0, which was relatively simple (block 2 determined the execution time). After the replication, blocks 0 and 2 are roughly equal

in complexity and one or the other will determine the execution time. This model will also compile and give the same results, the only difference being that it ports seamlessly to the PFP.

As noted in the comments of missil4.csl, there is a simple way to make a slight improvement in execution time, left as an exercise for the reader.

4.3 CHEATING TIME

One may think that the parallel implementation is not "correct" in the same sense that the serial implementation is correct, since it relies on "old" data that can be made available to all processors at the start of each integration step. This is simply not the case! The parallel implementation is a perfect implementation of proper numerical integration techniques applied to continuous systems for the purpose of digital simulation. As such, it is correct in the same sense as any corresponding serial implementation and will yield exactly the same results, assuming that computational precision is held constant.

The reason for this is that the parallel model relies only upon knowledge of the state variables at the beginning of each timestep. State variables are those variables which are integrated and thus contain the entire "memory" of the system. Intermediate algebraic variables are calculated on each processor as needed, and sometimes these calculations are replicated, as noted above, to eliminate the need for complex staging of communications.

It is possible, however, to force a little more parallelism into a model by allowing the use of "old" data. This will almost invariably affect the simulation results, so it must be done with discretion. The best candidates for this trick are sections of code which are fairly complex, generating intermediate variables which change relatively slowly over time. As an example, we will choose the sections of our missile model which calculate atmospheric damping and aerodynamic coefficients.

These sections are pulled out of the same lines of code which were replicated earlier on both block 0 and block 2. Since they contribute to the heavy processing on these blocks, they are reasonable candidates for pulling into separate blocks. It also seems reasonable that these coefficients would not change too much from one time step to the next.

The trick is shown in missil5.csl, in which two new blocks (4 and 5) are created. Each generates some new "derivative" variables, which are always set to zero. Then, when the "integration" is performed on the false state variables (C, CD, and Q), their values do not change from what they were set to inside the block. The result of this is that any block which needs C, CD, or Q gets their values one step late, but the calculation of these variables takes place in parallel.

This model also compiles and runs, but the results are different, as expected. If one plots the missile positions and velocities, there are no obvious differences, but listings of actual values for some variables show that the new model is not identical. When there is any doubt as to the

acceptability of these deviations, this "trick" should not be implemented, and model state variables should correspond to true state variables in a physical sense.

4.4 REAPING THE BENEFITS

It is possible, though, to keep the original model and still reap many more benefits from parallel processing. This is accomplished by taking this simple model and gradually adding more components. Most such components include their own state variables and depend only on other state variables. A seeker model, for example, would generate some representation of what the seeker is seeing (the seeker state) based upon the relative positions of the target and the interceptor, which are also states. Since there is invariably some delay between the actual imaging process and the output of seeker data, there is no problem associated with having to use the current position states to determine the "next" seeker state. In fact, the delay is normally much longer than the integration step, so the programmer would most likely deliberately insert more delay in order to make a good seeker model.

Another example of a component which can be easily added is an IMU, which generates estimated missile states, generally based on current accelerometer states (which are not the actual accelerations).

Reasonable seeker and IMU models can be added to the example with absolutely no increase in execution time, since they can be performed in parallel on two or more additional processors.

A more difficult subsystem to add would be the accelerometers themselves, which must generate the required acceleration estimates for the IMU. The reason for this is that they are most easily modeled as a procedure which generates acceleration estimates based on actual accelerations, but actual accelerations are not states. (In this discussion, we are referring only to rotational accelerations and velocities, of course.) If we arbitrarily declare the actual accelerations to be states and then use them as inputs to an accelerometer block, then we are introducing a delay of one integration time step, which does not accurately model the real system. While this may be tolerable, it is more reasonable to simply add the accelerometer model to the same block (or blocks) which calculate the true accelerations. This block would then output estimated accelerations as new states, in addition to whatever states it already generated. (Previously, this block probably just integrated the actual accelerations to generate the actual velocities, which it outputted as states. Now it would output both velocities and estimated accelerations.)

In a complex missile model, there will be many other systems which can be added as new parallel blocks, like the seeker and IMU above. There will also be a few like the accelerometers, which must be incorporated into existing blocks in order to maintain an accurate model. If certain blocks grow to the point where they force execution time to grow to an unacceptable level, it may be possible to partition them by other techniques. The emphasis here is on a basic way of partitioning the major elements in such a way that a serial implementation can be transferred almost effortlessly to a parallel implementation which can be fine-tuned later, if necessary.

5. SUMMARY

A proposal has been made that all simulations intended to be run on the PFP in the near future be written in ACSL, with some specific guidelines for structure. This has several advantages, including:

1. Specific identification of parallelism
2. Automatic translation to the PFP
3. Migration path to ADA
4. Built-in integration methods (in ACSL and on the PFP)
5. Support for specialized simulation functions (also in ACSL and on the PFP).

APPENDIX A: MISSIL.CSL (INITIAL PROGRAM)

```
PROGRAM - MISSILE AIRFRAME MODEL

      "----------------------A GENERIC MISSILE AIRFRAME MODEL IS "
      " DEVELOPED USING VECTORS FOR ALL THREE DIMENSIONAL QUANTITIES. "
      " THIS MODEL WILL RESPOND TO FIN DEFLECTIONS SO REPRESENTING THE "
      " OPEN LOOP AIRFRAME RESPONSE AND NEEDS A SEEKER, AUTOPILOT, "
      " ACTUATOR, MOTOR AND TARGET MODULE IN ORDER TO EVALUATE GUIDANCE"
      " EFFECTIVENESS "

INITIAL

      ALGORITHM       IALG = 4
      MAXTERVAL       MAXT = 0.010
      NSTEPS          NSTP = 1
      CINTERVAL       CINT = 0.020

      "----------------------SET UP IN CASE DICTIONARY REQUIRED "
      LOGICAL         DICTDM          $ CONSTANT DICTDM = .FALSE.
      IF(DICTDM) CALL LISTD(5)
      DICTDM  = .FALSE.
      "----------------------PASS STABILITY DERIVATIVE MATRIX TO THE "
      "                      COEFFICIENT GENERATION SUBROUTINE "
      CALL INIT(A)

END $" OF INITIAL "

DYNAMIC

DERIVATIVE

      "----------------------ENVIRONMENT MODULE "

      "----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      CONSTANT        G = 32.2
      "----------------------VELOCITY OF SOUND - FUNCTION OF ALTITUDE "
      TABLE           VS, 1, 10    ...
                    / 0.0     , 1.0E4   , 2.0E4   , 3.0E4   , 4.0E4   ...
                    , 5.0E4   , 6.0E4   , 7.0E4   , 8.0E4   , 9.0E4   ...
                    , 1186.5  , 1077.4  , 1036.4  , 994.8   , 968.1   ...
                    , 968.1   , 968.1   , 970.9   , 977.6   , 984.3   /
      "----------------------LOG OF ATMOSPHERIC DENSITY "
      TABLE           LRO, 1, 10   ...
                    / 0.0     , 1.0E4   , 2.0E4   , 3.0E4   , 4.0E4   ...
                    , 5.0E4   , 6.0E4   , 7.0E4   , 8.0E4   , 9.0E4   ...
                    ,-6.04191 ,-6.34502 ,-6.67084 ,-7.02346 ,-7.43995 ...
                    ,-7.91851 ,-8.39664 ,-8.87953 ,-9.36448 ,-9.87239/

      "----------------------CALCULATE ACTUAL ATMOSPHERIC DENSITY "
      RO      = EXP(LRO(RM(2)))

      "----------------------MISSILE AIRFRAME MODULE "

      "----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      REAL            ME(9), VMM(3), NM(3), NME(3), DL(4), CD(3), C(6)
      REAL            VM(3), VMD(3), VMIC(3), RM(3), RMD(3), RMIC(3)
      REAL            WM(3), WMD(3), WMIC(3), A(30)

      "----------------------MISSILE DIMENSIONAL CONSTANTS "
      CONSTANT        B = 3.95            , CBAR = 5.62
      CONSTANT        S = 13.9            , DXREF = 9.60
      CONSTANT        DL = 4*0.0
      "----------------------INITIAL CONDITION VALUES "
      CONSTANT        SIMIC = 0.0         , THMIC = 0.0
      CONSTANT        FIMIC = 0.0         , WMIC  = 3*0.0
      CONSTANT        VMIC = 2154.8, 2*0.0
      CONSTANT        RMIC = 0.0, 10000.0, 0.0
      "----------------------DEFINE ELEMENTS OF STABILITY DERIVATIVE "
      " MATRIX. LINEAR AERODATA IS ASSUMED FOR SIMPLICITY IN SUBROUTINE"
      " COEFF. NON-LINEAR AERODATA MAY BE INCORPORATED BY REWRITING "
      " THIS SUBROUTINE "
      CONSTANT        A =        ...
           0.148    , 0.0      , 0.0     , 0.0     , 0.0     , 0.0     ...
         , 0.0      ,-0.26     , 0.0     , 0.0     , 0.0     ,-0.286   ...
         , 0.0      , 0.0      ,-0.26    , 0.0     , 0.286   , 0.0     ...
         , 0.0      , 0.528    , 0.0     , 0.0     , 0.0     , 2.0     ...
         , 0.0      , 0.0      , 0.528   , 0.0     ,-2.0     , 0.0
      "----------------------ROLL DAMPING - FUNCTION OF MACH NUMBER "
      TABLE           CLP, 1, 5    ...
                    / 0.0     , 0.8     , 1.0     , 1.2     , 2.0     ...
                    ,-0.21    ,-0.21    ,-0.20    ,-0.19    ,-0.18    /
      "----------------------PITCH DAMPING - FUNCTION OF MACH NUMBER "
      TABLE           CMQ, 1, 5    ...
```

```
                  / 0.0      , 0.8      , 1.0      , 1.2      , 2.0      ...
                  ,-3.8      ,-2.0      ,-1.5      ,-2.0      ,-2.1     /

      "-----------------------MAGNITUDE OF MISSILE VELOCITY "
      MVM     = SQRT(DOT(VM, VM))
      "-----------------------MAKE *ME* MATRIX FROM ORIENTATION ANGLES "
      CALL MMK(ME = FIM, 1, THM, 3, SIM, 2)
      "-----------------------ROTATE VELOCITY TO MISSILE FRAME "
      CALL VECROT(VMM = VM, ME)
      "-----------------------LATERAL AND VERTICAL ANGLES OF ATTACK "
      AL2     = ATAN(-VMM(3)/VMM(1))
      AL3     = ATAN( VMM(2)/VMM(1))
      "-----------------------MACH NUMBER AND DYNAMIC PRESSURE "
      MACH    = MVM/VS(RM(2))
      Q       = 0.5*RO*MVM**2
      "-----------------------CALCULATE DAMPING DERIVATIVES "
     PROCEDURAL(CD = MVM, MACH, WM)
      CD(1)   = 0.5*CLP(MACH)*B*WM(1)/MVM
      CCVV    = 0.5*CMQ(MACH)*CBAR/MVM
      CD(2)   = CCVV*WM(2)
      CD(3)   = CCVV*WM(3)
     END $" OF PROCEDURAL "
      "-----------------------GET MOMENTS AND FORCE AERO COEFFICIENTS "
      " AND CORRECT LATERAL MOMENTS FOR SHIFT IN CENTRE OF GRAVITY "
      " POSITION "
     PROCEDURAL(C = AL2, AL3, DL, MACH, DXCG, DXREF)
      CALL COEFF(C = AL2, AL3, DL, MACH)
      C(2)    = C(2) - (DXCG - DXREF)*C(6)/CBAR
      C(3)    = C(3) + (DXCG - DXREF)*C(5)/CBAR
     END $" OF PROCEDURAL "
      "-----------------------CALCULATE ACCELERATION DUE TO AERODYNAMIC"
      "                       EFFECTS AND ROTATION RATE DERIVATIVES "
     PROCEDURAL(NM, WMD = Q, C, CD, WM, MASS, IXX, IYY)
      NM(1)   = (Q*S*C(4) + THRUST)/MASS
      NM(2)   = Q*S*C(5)/MASS
      NM(3)   = Q*S*C(6)/MASS
      WMD(1)  = Q*S*B*(C(1) + CD(1))/IXX
      WMD(2)  = Q*S*CBAR*(C(2) + CD(2))/IYY + WM(1)*WM(3)
      WMD(3)  = Q*S*CBAR*(C(3) + CD(3))/IYY - WM(1)*WM(2)
     END $" OF PROCEDURAL "
      "-----------------------ROTATE ACCELERATION VECTOR TO EARTH FRAME"
      CALL INVROT(NME = NM, ME)
      "-----------------------CALCULATE VELOCITY DERIVATIVES IN THE  "
      "                       EARTH FRAME - NEEDS GRAVITY ADDING IN "
     PROCEDURAL(VMD = NME, G)
      VMD(1)  = NME(1)
      VMD(2)  = NME(2) - G
      VMD(3)  = NME(3)
     END $" OF PROCEDURAL "
      "-----------------------YAW ANGLE DERIVATIVE "
      SIMD    = (WM(2)*COS(FIM) - WM(3)*SIN(FIM))/COS(THM)
      "-----------------------INTEGRATE FOR ALL EULER ANGLES - NOTE USE"
      "                       OF VECTOR INTEGRATOR FOR SINGLE ELEMENT "
      SIM     = INTVC(SIMD, SIMIC)
      THM     = INTEG(WM(2)*SIN(FIM) + WM(3)*COS(FIM), THMIC)
      FIM     = INTEG(WM(1) - SIMD*SIN(THM), FIMIC)
      "-----------------------VECTOR INTEGRATE FOR ROTATIONAL VELOCITY "
      WM      = INTVC(WMD, WMIC)
      "-----------------------TRANSLATIONAL VELOCITY "
      VM      = INTVC(VMD, VMIC)
      "-----------------------TRANSLATIONAL POSITION - NOTE THE DERIV- "
      " ATIVE VECTOR CANNOT BE A STATE VECTOR (VELOCITY) AS WELL "
      CALL XFERB(RMD = VM, 3)
      RM      = INTVC(RMD, RMIC)

      "-----------------------MOTOR MODULE "

      "-----------------------SIMPLE VERSION WITH ZERO THRUST SPECIF- "
      "                       YING A BURNT OR GLIDE CONDITION "
      CONSTANT       THRUST = 0.0       , MASS = 8.77
      CONSTANT       IXX = 8.77         , IYY = 361.8
      CONSTANT       DXCG = 10.2

END $" OF DERIVATIVE "

      "-----------------------STOP ON ELAPSED TIME "
      CONSTANT       TSTP = 1.99
      TERMT(T .GE. TSTP)

END $" OF DYNAMIC "

END $" OF PROGRAM "
      SUBROUTINE INIT(C)
C----------------------------FORTRAN SUBROUTINE WHOSE ONLY JOB IS TO
C     TRANSFER THE STABILITY DERIVATIVE MATRIX TO AN ARRAY IN LABELLED
C     COMMON SO THAT IT MAY BE ACCESSED IN SUBROUTINE COEFF. NOTE NO
```

```
C     COMMON BLOCKS MAY BE DEFINED IN THE ACSL MODEL DEFINITION SECTION
C
      COMMON/STABD/  A(6,5)
      DATA           LENGTH  / 30     /
C
C----------------------------TRANSFER BLOCK
      CALL XFERB(C, LENGTH, A)
      RETURN
C
      END
      SUBROUTINE COEFF(AL2, AL3, DL, MACH, C)
C----------------------------COMPUTES SIX AERODYNAMIC COEFFICIENTS -
C     THREE MOMENTS, C(1), C(2) AND C(3), AND THREE FORCES, C(4), C(5)
C     AND C(6). MOMENTS ARE ABOUT AXES CENTRED AT THE REFERENCE POINT
C     AND MUST BE CORRECTED FOR CENTRE OF GAVITY SHIFT.
C
C     INPUTS
C
C     AL2      ANGLE OF ATTACK ABOUT *M2* - POSITIVE WIND FROM LEFT
C     AL3      ANGLE OF ATTACK ABOUT *M3* - POSITIVE WIND FROM ABOVE
C     DL       ARRAY OF FOUR FIN DEFLECTIONS
C     MACH     MACH NUMBER (REAL)
C
C     OUTPUTS
C
C     C        ARRAY OF SIX AERODYNAMIC COEFFICIENTS
C
      REAL           DL(4)    , C(6)    , MACH
C
      COMMON/STABD/  A(6,5)
C
C----------------------------COMPUTE EQUIVALENT CONTROL SURFACE DEFL-
C                            ECTIONS FROM THE FOUR SURFACE ANGLES
      DLA      = 0.25*(DL(3) + DL(4) - DL(1) - DL(2))
      DLY      = 0.50*(DL(1) + DL(3))
      DLZ      = 0.50*(DL(2) + DL(4))
C----------------------------COMPUTE EACH MOMENT ASSUMING IT IS LINEAR
C                            IN EACH OF THE ARGUMENTS
      DO 110 J = 1, 6
      C(J)     = A(J,1)*DLA + A(J,2)*DLY + A(J,3)*DLZ + A(J,4)*AL2
     .                + A(J,5)*AL3
  110 CONTINUE
      RETURN
C
      END

      FUNCTION DOT (A, B)
C----------------------------COMPUTE VECTOR DOT PRODUCT OF TWO VECTORS.
C
C     PROGRAMMER V. B. WAYLAND
C
C     INPUTS
C
C     A AND B ARE ARRAYS OF LENGTH 3
C
C     OUTPUT
C
C     DOT IS A REAL FUNCTION RETURNING THE DOT PRODUCT OF A  AND B
C
      DIMENSION      A(3)    , B(3)
C
      DOT    = A(1) * B(1)  +  A(2) * B(2)  +  A(3) * B(3)
      RETURN
C
      END
      SUBROUTINE INV ROT(VIN, RMX, VOUT)
C----------------------------INVERSE VECTOR ROTATION. INVERSELY ROTATE
C     AN INPUT VECTOR, VIN, FROM ONE COORDINATE SYSTEM THRU A TRANSPOSED
C     ROTATION MATRIX, RMX. THE NEW VECTOR IS VOUT.
C
C     B = (AB)T * A
C
C     INPUT
C
C     VIN       INPUT 3 VECTOR
C     RMX       3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT      OUTPUT 3 VECTOR
C
      DIMENSION      VIN(3)   , RMX(3,3), VOUT(3)
C
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(2,1)*VIN(2) + RMX(3,1)*VIN(3)
      VOUT(2) = RMX(1,2)*VIN(1) + RMX(2,2)*VIN(2) + RMX(3,2)*VIN(3)
      VOUT(3) = RMX(1,3)*VIN(1) + RMX(2,3)*VIN(2) + RMX(3,3)*VIN(3)
```

```
      RETURN
C
      END
      SUBROUTINE  MMK(A, NA, B, NB, C, NC, RM)
C     1A-0     30 DEC 68       MAKE A DIRECTION COSINE MATRIX
C----------------------------ROUTINE GENERATES A DIRECTION COSINE MATRX
C     BY ROTATING IN ORDER
C
C     1)ANGLE C ABOUT THE NC AXIS
C     2)ANGLE B ABOUT THE NB AXIS
C     3)ANGLE A ABOUT THE NA AXIS
C
C     INPUTS
C
C     ANGLES A, B, C IN RADIANS
C     NA, NB, NC - A NUMBER BETWEEN 1 AND 3 CORRESPONDING TO AXIS
C                  ABOUT WHICH EACH ANGLE IS ROTATED
C
C     OUTPUT
C
C     RM -- A 3X3 DIRECTION COSINE MATRIX
C
      REAL            AM(3,3) , BM(3,3) , CM(3,3) , RM(3,3)
      REAL            T(9)
C
C     NOTE FOR FORMING A DIRECTION COSINE MATRIX FROM EULER ANGLES THE
C     CONVENTION IS TO ROTATE ANGLE PHI   ABOUT THE NO. 1 AXIS, ANGLE
C     PSI ABOUT THE NO. 2 AXIS AND ANGLE THETA ABOUT THE NO. 3 AXIS
C
C----------------------------GENERATE THE ROTATION MATRIX FOR EACH ANG.
      CALL ROTMX ( A, NA, AM)
      CALL ROTMX ( B, NB, BM)
      CALL ROTMX ( C, NC, CM)
C----------------------------MATRIX MULTIPLY THE INTERMEDIATE MATRICES
      CALL  MML XY(BM,CM,T)
      CALL  MML XY(AM,T,RM)
      RETURN
C
      END
      SUBROUTINE  MML XY (X, Y, Z)
C----------------------------MATRIX MULTIPLY ROUTINES FOR TWO 3X3
C     MATRICES. FIRST ENTRY CONTAINS NO TRANSPOSES
C
C     Z = (X) * (Y)
C
C     INPUT
C
C     X          FIRST 3X3 MATRIX
C     Y          SECOND 3X3 MATRIX
C
C     OUTPUT
C
C     Z          RESULTING 3X3 MATRIX WHERE
C     Z(I,J)  = X(I,1)*Y(1,J) + X(I,2)*Y(2,J) + X(I,3)*Y(3,J)
C
      DIMENSION      X(3,3)  ,  Y(3,3) , Z(3,3)
C
      Z(1,1)  = X(1,1)*Y(1,1) + X(1,2)*Y(2,1) + X(1,3)*Y(3,1)
      Z(2,1)  = X(2,1)*Y(1,1) + X(2,2)*Y(2,1) + X(2,3)*Y(3,1)
      Z(3,1)  = X(3,1)*Y(1,1) + X(3,2)*Y(2,1) + X(3,3)*Y(3,1)
      Z(1,2)  = X(1,1)*Y(1,2) + X(1,2)*Y(2,2) + X(1,3)*Y(3,2)
      Z(2,2)  = X(2,1)*Y(1,2) + X(2,2)*Y(2,2) + X(2,3)*Y(3,2)
      Z(3,2)  = X(3,1)*Y(1,2) + X(3,2)*Y(2,2) + X(3,3)*Y(3,2)
      Z(1,3)  = X(1,1)*Y(1,3) + X(1,2)*Y(2,3) + X(1,3)*Y(3,3)
      Z(2,3)  = X(2,1)*Y(1,3) + X(2,2)*Y(2,3) + X(2,3)*Y(3,3)
      Z(3,3)  = X(3,1)*Y(1,3) + X(3,2)*Y(2,3) + X(3,3)*Y(3,3)
      RETURN
C
      END
      SUBROUTINE  ROTMX( X, I, XM)
C     1A-0     30 DEC 68       ROTATION MATRIX
C----------------------------GENERATE BY STARTING WITH AN IDENTITY MATX
C     PUT THE COSINE OF ANGLE X ON THE DIAGONAL AND +SIN(X) AND -SIN(X)
C     ON OFF DIAGONALS
C
      REAL           XM(3,3)
      INTEGER        II T(3), III T(3)
C
      DATA     II T / 2 , 3  , 1  /
     .        ,III T/ 3 , 1  , 2  /
C
      SX      = SIN(X)
      CX      = COS(X)
      II      = II T(I)
      III     = III T(I)
C
```

```
      XM(I,I) = 1.0
      XM(I,II)= 0.0
      XM(II,I)= 0.0
      XM(I,III)= 0.0
      XM(III,I)= 0.0
C
      XM(II,II)= CX
      XM(III,III)= CX
      XM(II,III) =  SX
      XM(III,II) = -SX
C
      RETURN
C
      END
      SUBROUTINE  VEC ROT (VIN, RMX, VOUT)
C     1A-0   25 NOV 68        VECTOR ROTATION
C-----------------------------ROTATE AN INPUT VECTOR, VIN, FROM ONE
C     COORDINATE SYSTEM THRU A ROTATION MATRIX, RMX.  THE NEW VECTOR IS
C     VOUT.
C
C     A = (AB) * B
C
C     INPUT
C
C     VIN       INPUT 3 VECTOR
C     RMX       3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT      OUTPUT 3 VECTOR
C
      DIMENSION      VIN(3)   , RMX(3,3), VOUT(3)
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(1,2)*VIN(2) + RMX(1,3)*VIN(3)
      VOUT(2) = RMX(2,1)*VIN(1) + RMX(2,2)*VIN(2) + RMX(2,3)*VIN(3)
      VOUT(3) = RMX(3,1)*VIN(1) + RMX(3,2)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
```

APPENDIX B: MISSIL2.CSL

```
PROGRAM - MISSILE AIRFRAME MODEL

      " Modified from missil.csl. "
      " Simply defined four procedural blocks corresponding to major "
      " functional elements within the model.  Moved some lines of "
      " code to fit the block structure, but did not change any "
      " ordering of statements that were already in procedural blocks."
      " Once all four blocks were defined, input/output dependencies "
      " were explicitly given in procedural statement, and the "
      " original procedurals were deleted (ACSL does not allow nested "
      " procedurals). "
      " This version was not expected to translate, since there is no"
      " way to sort the statements. (Note that NM depends on C, CD,"
      " and Q, and that C, CD, and Q depend on NM.  This circular re-"
      " lationship is not allowed. "

      "-----------------------A GENERIC MISSILE AIRFRAME MODEL IS "
      " DEVELOPED USING VECTORS FOR ALL THREE DIMENSIONAL QUANTITIES. "
      " THIS MODEL WILL RESPOND TO FIN DEFLECTIONS SO REPRESENTING THE "
      " OPEN LOOP AIRFRAME RESPONSE AND NEEDS A SEEKER, AUTOPILOT, "
      " ACTUATOR, MOTOR AND TARGET MODULE IN ORDER TO EVALUATE GUIDANCE"
      " EFFECTIVENESS "

INITIAL

      ALGORITHM      IALG = 4
      MAXTERVAL      MAXT = 0.010
      NSTEPS         NSTP = 1
      CINTERVAL      CINT = 0.020

      "-----------------------SET UP IN CASE DICTIONARY REQUIRED "
      LOGICAL        DICTDM            $ CONSTANT DICTDM = .FALSE.
      IF(DICTDM) CALL LISTD(5)
      DICTDM  = .FALSE.
      "-----------------------PASS STABILITY DERIVATIVE MATRIX TO THE "
      "                       COEFFICIENT GENERATION SUBROUTINE "
      CALL INIT(A)

END $" OF INITIAL "

DYNAMIC

DERIVATIVE

      "-----------------------ENVIRONMENT MODULE "

      "-----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      CONSTANT       G = 32.2
      "-----------------------VELOCITY OF SOUND - FUNCTION OF ALTITUDE "
      TABLE          VS, 1, 10    ...
                   / 0.0      , 1.0E4    , 2.0E4    , 3.0E4    , 4.0E4    ...
                   , 5.0E4    , 6.0E4    , 7.0E4    , 8.0E4    , 9.0E4    ...
                   , 1186.5   , 1077.4   , 1036.4   , 994.8    , 968.1    ...
                   , 968.1    , 968.1    , 970.9    , 977.6    , 984.3  /
      "-----------------------LOG OF ATMOSPHERIC DENSITY "
      TABLE          LRO, 1, 10   ...
                   / 0.0      , 1.0E4    , 2.0E4    , 3.0E4    , 4.0E4    ...
                   , 5.0E4    , 6.0E4    , 7.0E4    , 8.0E4    , 9.0E4    ...
                   ,-6.04191 ,-6.34502 ,-6.67084 ,-7.02346 ,-7.43995 ...
                   ,-7.91851 ,-8.39664 ,-8.87953 ,-9.36448 ,-9.87239/

      "-----------------------MISSILE AIRFRAME MODULE "

      "-----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      REAL           ME(9), VMM(3), NM(3), NME(3), DL(4), CD(3), C(6)
      REAL           VM(3), VMD(3), VMIC(3), RM(3), RMD(3), RMIC(3)
      REAL           WM(3), WMD(3), WMIC(3), A(30)

      "-----------------------MISSILE DIMENSIONAL CONSTANTS "
      CONSTANT       B = 3.95            , CBAR = 5.62
      CONSTANT       S = 13.9            , DXREF = 9.60
      CONSTANT       DL = 4*0.0
      "-----------------------INITIAL CONDITION VALUES "
      CONSTANT       SIMIC = 0.0         , THMIC = 0.0
      CONSTANT       FIMIC = 0.0         , WMIC  = 3*0.0
      CONSTANT       VMIC = 2154.8, 2*0.0
      CONSTANT       RMIC = 0.0, 10000.0, 0.0
      "-----------------------DEFINE ELEMENTS OF STABILITY DERIVATIVE "
      " MATRIX. LINEAR AERODATA IS ASSUMED FOR SIMPLICITY IN SUBROUTINE"
      " COEFF. NON-LINEAR AERODATA MAY BE INCORPORATED BY REWRITING "
      " THIS SUBROUTINE "
```

```
CONSTANT        A =     ...
     0.148    , 0.0     , 0.0     , 0.0     , 0.0     , 0.0      ...
    , 0.0     ,-0.26    , 0.0     , 0.0     , 0.0     ,-0.286    ...
    , 0.0     , 0.0     ,-0.26    , 0.0     , 0.286   , 0.0      ...
    , 0.0     , 0.528   , 0.0     , 0.0     , 0.0     , 2.0      ...
    , 0.0     , 0.0     , 0.528   , 0.0     ,-2.0     , 0.0
"-----------------------ROLL DAMPING - FUNCTION OF MACH NUMBER "
TABLE           CLP, 1, 5   ...
              / 0.0     , 0.8     , 1.0     , 1.2     , 2.0      ...
              ,-0.21    ,-0.21    ,-0.20    ,-0.19    ,-0.18    /
"-----------------------PITCH DAMPING - FUNCTION OF MACH NUMBER "
TABLE           CMQ, 1, 5   ...
              / 0.0     , 0.8     , 1.0     , 1.2     , 2.0      ...
              ,-3.8     ,-2.0     ,-1.5     ,-2.0     ,-2.1     /


"block 0 : motor, aerodynamics, and rotational velocity dynamics"
   procedural(WMD, NM = CD, C, Q, WM)
    ZBLOCK=0

    "-----------------------MOTOR MODULE "

    "-----------------------SIMPLE VERSION WITH ZERO THRUST SPECIF- "
    "                       YING A BURNT OR GLIDE CONDITION "
    CONSTANT       THRUST = 0.0       , MASS = 8.77
    CONSTANT       IXX = 8.77         , IYY = 361.8
    CONSTANT       DXCG = 10.2


    "-----------------------CALCULATE ACCELERATION DUE TO AERODYNAMIC"
    "                       EFFECTS AND ROTATION RATE DERIVATIVES "
    NM(1)   = (Q*S*C(4) + THRUST)/MASS
    NM(2)   = Q*S*C(5)/MASS
    NM(3)   = Q*S*C(6)/MASS
    WMD(1)  = Q*S*B*(C(1) + CD(1))/IXX
    WMD(2)  = Q*S*CBAR*(C(2) + CD(2))/IYY + WM(1)*WM(3)
    WMD(3)  = Q*S*CBAR*(C(3) + CD(3))/IYY - WM(1)*WM(2)
   end $"of procedural"

    "-----------------------VECTOR INTEGRATE FOR ROTATIONAL VELOCITY "
    WM      = INTVC(WMD, WMIC)
"end of block 0"


"block 1 : rotational posn dynamics"
   procedural(SIMD = WM, THM, FIM )
    ZBLOCK=1
    "-----------------------YAW ANGLE DERIVATIVE "
    SIMD    = (WM(2)*COS(FIM) - WM(3)*SIN(FIM))/COS(THM)

   end $"of procedural"

    "-----------------------INTEGRATE FOR ALL EULER ANGLES - NOTE USE"
    "                       OF VECTOR INTEGRATOR FOR SINGLE ELEMENT "
    SIM     = INTVC(SIMD, SIMIC)
    THM     = INTEG(WM(2)*SIN(FIM) + WM(3)*COS(FIM), THMIC)
    FIM     = INTEG(WM(1) - SIMD*SIN(THM), FIMIC)
"end of block 1"


"block 2 : translational velocity dynamics"
   procedural(VMD,Q,CD,C,ME = NM,FIM,SIM,THM,RM,WM,VM )
    ZBLOCK=2
    "-----------------------MAKE *ME* MATRIX FROM ORIENTATION ANGLES "
    CALL MMK(ME = FIM, 1, THM, 3, SIM, 2)
    "-----------------------CALCULATE ACTUAL ATMOSPHERIC DENSITY "
    RO      = EXP(LRO(RM(2)))

    "-----------------------MAGNITUDE OF MISSILE VELOCITY "
    MVM     = SQRT(DOT(VM, VM))
    "-----------------------ROTATE VELOCITY TO MISSILE FRAME "
    CALL VECROT(VMM = VM, ME)
    "-----------------------LATERAL AND VERTICAL ANGLES OF ATTACK "
    AL2     = ATAN(-VMM(3)/VMM(1))
    AL3     = ATAN( VMM(2)/VMM(1))
    "-----------------------MACH NUMBER AND DYNAMIC PRESSURE "
    MACH    = MVM/VS(RM(2))
    Q       = 0.5*RO*MVM**2
    "-----------------------CALCULATE DAMPING DERIVATIVES "
    CD(1)   = 0.5*CLP(MACH)*B*WM(1)/MVM
    CCVV    = 0.5*CMQ(MACH)*CBAR/MVM
    CD(2)   = CCVV*WM(2)
    CD(3)   = CCVV*WM(3)
```

```
        "-----------------------GET MOMENTS AND FORCE AERO COEFFICIENTS "
        " AND CORRECT LATERAL MOMENTS FOR SHIFT IN CENTRE OF GRAVITY "
        " POSITION "
        CALL COEFF(C = AL2, AL3, DL, MACH)
        C(2)    = C(2) - (DXCG - DXREF)*C(6)/CBAR
        C(3)    = C(3) + (DXCG - DXREF)*C(5)/CBAR
        "-----------------------ROTATE ACCELERATION VECTOR TO EARTH FRAME"
        CALL INVROT(NME = NM, ME)
        "-----------------------CALCULATE VELOCITY DERIVATIVES IN THE  "
        "                       EARTH FRAME - NEEDS GRAVITY ADDING IN "
        VMD(1)  = NME(1)
        VMD(2)  = NME(2) - G
        VMD(3)  = NME(3)
       end $" of procedual "
        "-----------------------TRANSLATIONAL VELOCITY "
        VM      = INTVC(VMD, VMIC)
"end of block 2"


"block 3 : translational position dynamics"
       procedural(RMD = VM)
        ZBLOCK=3
        "-----------------------TRANSLATIONAL POSITION - NOTE THE DERIV- "
        " ATIVE VECTOR CANNOT BE A STATE VECTOR (VELOCITY) AS WELL "
        CALL XFERB(RMD = VM, 3)
       end $" of procedural "
        RM      = INTVC(RMD, RMIC)
"end of block 3"


END $" OF DERIVATIVE "

        "-----------------------STOP ON ELAPSED TIME "
        CONSTANT        TSTP = 1.99
        TERMT(T .GE. TSTP)

END $" OF DYNAMIC "

END $" OF PROGRAM "
      SUBROUTINE INIT(C)
C-----------------------------FORTRAN SUBROUTINE WHOSE ONLY JOB IS TO
C     TRANSFER THE STABILITY DERIVATIVE MATRIX TO AN ARRAY IN LABELLED
C     COMMON SO THAT IT MAY BE ACCESSED IN SUBROUTINE COEFF. NOTE NO
C     COMMON BLOCKS MAY BE DEFINED IN THE ACSL MODEL DEFINITION SECTION
C
      COMMON/STABD/  A(6,5)
      DATA           LENGTH  / 30     /
C
C-----------------------------TRANSFER BLOCK
      CALL XFERB(C, LENGTH, A)
      RETURN
C
      END
      SUBROUTINE COEFF(AL2, AL3, DL, MACH, C)
C-----------------------------COMPUTES SIX AERODYNAMIC COEFFICIENTS -
C     THREE MOMENTS, C(1), C(2) AND C(3), AND THREE FORCES, C(4), C(5)
C     AND C(6). MOMENTS ARE ABOUT AXES CENTRED AT THE REFERENCE POINT
C     AND MUST BE CORRECTED FOR CENTRE OF GRAVITY SHIFT.
C
C     INPUTS
C
C     AL2       ANGLE OF ATTACK ABOUT *M2* - POSITIVE WIND FROM LEFT
C     AL3       ANGLE OF ATTACK ABOUT *M3* - POSITIVE WIND FROM ABOVE
C     DL        ARRAY OF FOUR FIN DEFLECTIONS
C     MACH      MACH NUMBER (REAL)
C
C     OUTPUTS
C
C     C         ARRAY OF SIX AERODYNAMIC COEFFICIENTS
C
      REAL           DL(4)   , C(6)    , MACH
C
      COMMON/STABD/  A(6,5)
C
C-----------------------------COMPUTE EQUIVALENT CONTROL SURFACE DEFL-
C                             ECTIONS FROM THE FOUR SURFACE ANGLES
      DLA     = 0.25*(DL(3) + DL(4) - DL(1) - DL(2))
      DLY     = 0.50*(DL(1) + DL(3))
      DLZ     = 0.50*(DL(2) + DL(4))
C-----------------------------COMPUTE EACH MOMENT ASSUMING IT IS LINEAR
C                             IN EACH OF THE ARGUMENTS
      DO 110 J = 1, 6
      C(J)    = A(J,1)*DLA + A(J,2)*DLY + A(J,3)*DLZ + A(J,4)*AL2
     .                     + A(J,5)*AL3
```

```
  110 CONTINUE
      RETURN
C
      END
      FUNCTION DOT (A, B)
C----------------------------COMPUTE VECTOR DOT PRODUCT OF TWO VECTORS.
C
C     PROGRAMMER V. B. WAYLAND
C
C     INPUTS
C
C     A AND B ARE ARRAYS OF LENGTH 3
C
C     OUTPUT
C
C     DOT IS A REAL FUNCTION RETURNING THE DOT PRODUCT OF A  AND B
C
      DIMENSION      A(3)    , B(3)
C
      DOT     = A(1) * B(1)  +  A(2) * B(2)  +  A(3) * B(3)
      RETURN
C
      END
      SUBROUTINE INV ROT(VIN, RMX, VOUT)
C----------------------------INVERSE VECTOR ROTATION. INVERSELY ROTATE
C     AN INPUT VECTOR, VIN, FROM ONE COORDINATE SYSTEM THRU A TRANSPOSED
C     ROTATION MATRIX, RMX. THE NEW VECTOR IS VOUT.
C
C     B = (AB)T * A
C
C     INPUT
C
C     VIN       INPUT 3 VECTOR
C     RMX       3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT      OUTPUT 3 VECTOR
C
      DIMENSION      VIN(3)  , RMX(3,3), VOUT(3)
C
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(2,1)*VIN(2) + RMX(3,1)*VIN(3)
      VOUT(2) = RMX(1,2)*VIN(1) + RMX(2,2)*VIN(2) + RMX(3,2)*VIN(3)
      VOUT(3) = RMX(1,3)*VIN(1) + RMX(2,3)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
      SUBROUTINE  MMK(A, NA, B, NB, C, NC, RM)
C     1A-0     30 DEC 68        MAKE A DIRECTION COSINE MATRIX
C----------------------------ROUTINE GENERATES A DIRECTION COSINE MATRX
C     BY ROTATING IN ORDER
C
C     1)ANGLE C ABOUT THE NC AXIS
C     2)ANGLE B ABOUT THE NB AXIS
C     3)ANGLE A ABOUT THE NA AXIS
C
C     INPUTS
C
C     ANGLES A, B, C IN RADIANS
C     NA, NB, NC - A NUMBER BETWEEN 1 AND 3 CORRESPONDING TO AXIS
C                  ABOUT WHICH EACH ANGLE IS ROTATED
C
C     OUTPUT
C
C     RM -- A 3X3 DIRECTION COSINE MATRIX
C
      REAL           AM(3,3) , BM(3,3) , CM(3,3) , RM(3,3)
      REAL           T(9)
C
C     NOTE FOR FORMING A DIRECTION COSINE MATRIX FROM EULER ANGLES THE
C     CONVENTION IS TO ROTATE ANGLE PHI   ABOUT THE NO. 1 AXIS, ANGLE
C     PSI ABOUT THE NO. 2 AXIS AND ANGLE THETA ABOUT THE NO. 3 AXIS
C
C----------------------------GENERATE THE ROTATION MATRIX FOR EACH ANG.
      CALL ROTMX ( A, NA, AM)
      CALL ROTMX ( B, NB, BM)
      CALL ROTMX ( C, NC, CM)
C----------------------------MATRIX MULTIPLY THE INTERMEDIATE MATRICES
      CALL  MML XY(BM,CM,T)
      CALL  MML XY(AM,T,RM)
      RETURN
C
      END
      SUBROUTINE  MML XY (X, Y, Z)
C----------------------------MATRIX MULTIPLY ROUTINES FOR TWO 3X3
```

```
C     MATRICES. FIRST ENTRY CONTAINS NO TRANSPOSES
C
C     Z = (X) * (Y)
C
C     INPUT
C
C     X              FIRST 3X3 MATRIX
C     Y              SECOND 3X3 MATRIX
C
C     OUTPUT
C
C     Z              RESULTING 3X3 MATRIX WHERE
C     Z(I,J)   = X(I,1)*Y(1,J) + X(I,2)*Y(2,J) + X(I,3)*Y(3,J)
C
      DIMENSION      X(3,3)  ,  Y(3,3) , Z(3,3)
C
      Z(1,1)   = X(1,1)*Y(1,1) + X(1,2)*Y(2,1) + X(1,3)*Y(3,1)
      Z(2,1)   = X(2,1)*Y(1,1) + X(2,2)*Y(2,1) + X(2,3)*Y(3,1)
      Z(3,1)   = X(3,1)*Y(1,1) + X(3,2)*Y(2,1) + X(3,3)*Y(3,1)
      Z(1,2)   = X(1,1)*Y(1,2) + X(1,2)*Y(2,2) + X(1,3)*Y(3,2)
      Z(2,2)   = X(2,1)*Y(1,2) + X(2,2)*Y(2,2) + X(2,3)*Y(3,2)
      Z(3,2)   = X(3,1)*Y(1,2) + X(3,2)*Y(2,2) + X(3,3)*Y(3,2)
      Z(1,3)   = X(1,1)*Y(1,3) + X(1,2)*Y(2,3) + X(1,3)*Y(3,3)
      Z(2,3)   = X(2,1)*Y(1,3) + X(2,2)*Y(2,3) + X(2,3)*Y(3,3)
      Z(3,3)   = X(3,1)*Y(1,3) + X(3,2)*Y(2,3) + X(3,3)*Y(3,3)
      RETURN
C
      END
      SUBROUTINE  ROTMX( X, I, XM)
C     1A-0     30 DEC 68      ROTATION MATRIX
C---------------------------GENERATE BY STARTING WITH AN IDENTITY MATX
C     PUT THE COSINE OF ANGLE X ON THE DIAGONAL AND +SIN(X) AND -SIN(X)
C     ON OFF DIAGONALS
C
      REAL           XM(3,3)
      INTEGER        II T(3), III T(3)
C
      DATA      II T / 2 , 3  , 1  /
     .         ,III T/ 3 , 1  , 2  /
C
      SX       = SIN(X)
      CX       = COS(X)
      II       = II T(I)
      III      = III T(I)
C
      XM(I,I) = 1.0
      XM(I,II)= 0.0
      XM(II,I)= 0.0
      XM(I,III)= 0.0
      XM(III,I)= 0.0
C
      XM(II,II)= CX
      XM(III,III)= CX
      XM(II,III) =  SX
      XM(III,II) = -SX
C
      RETURN
C
      END
      SUBROUTINE  VEC ROT (VIN, RMX, VOUT)
C     1A-0   25 NOV 68          VECTOR ROTATION
C----------------------------ROTATE AN INPUT VECTOR, VIN, FROM ONE
C     COORDINATE SYSTEM THRU A ROTATION MATRIX, RMX.  THE NEW VECTOR IS
C     VOUT.
C
C     A = (AB) * B
C
C     INPUT
C
C     VIN       INPUT 3 VECTOR
C     RMX       3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT      OUTPUT 3 VECTOR
C
      DIMENSION      VIN(3)   , RMX(3,3), VOUT(3)
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(1,2)*VIN(2) + RMX(1,3)*VIN(3)
      VOUT(2) = RMX(2,1)*VIN(1) + RMX(2,2)*VIN(2) + RMX(2,3)*VIN(3)
      VOUT(3) = RMX(3,1)*VIN(1) + RMX(3,2)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
```

APPENDIX C: MISSIL3.CSL

```
PROGRAM - MISSILE AIRFRAME MODEL

      " Modified from missil2.csl "
      " split NM assignments out of block 1 and put them into "
      " block 2.  This eliminates circular definition that kept "
      " missil2 from translating.  Note that the NM statements "
      " were originally (in missil.csl) within a procedural block, "
      " but no dependencies were violated when they were moved to "
      " block 2.  This model runs under ACSL and its output is "
      " identical to that of missil.csl. "

      "----------------------A GENERIC MISSILE AIRFRAME MODEL IS "
      " DEVELOPED USING VECTORS FOR ALL THREE DIMENSIONAL QUANTITIES. "
      " THIS MODEL WILL RESPOND TO FIN DEFLECTIONS SO REPRESENTING THE "
      " OPEN LOOP AIRFRAME RESPONSE AND NEEDS A SEEKER, AUTOPILOT, "
      " ACTUATOR, MOTOR AND TARGET MODULE IN ORDER TO EVALUATE GUIDANCE"
      " EFFECTIVENESS "

INITIAL

      ALGORITHM      IALG = 4
      MAXTERVAL      MAXT = 0.010
      NSTEPS         NSTP = 1
      CINTERVAL      CINT = 0.020

      "----------------------SET UP IN CASE DICTIONARY REQUIRED "
      LOGICAL        DICTDM             $ CONSTANT DICTDM = .FALSE.
      IF(DICTDM) CALL LISTD(5)
      DICTDM  = .FALSE.
      "----------------------PASS STABILITY DERIVATIVE MATRIX TO THE "
      "                      COEFFICIENT GENERATION SUBROUTINE "
      CALL INIT(A)

END $" OF INITIAL "

DYNAMIC

DERIVATIVE

      "----------------------ENVIRONMENT MODULE "

      "----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      CONSTANT       G = 32.2
      "----------------------VELOCITY OF SOUND - FUNCTION OF ALTITUDE "
      TABLE          VS, 1, 10    ...
                   / 0.0     , 1.0E4   , 2.0E4   , 3.0E4   , 4.0E4    ...
                   , 5.0E4   , 6.0E4   , 7.0E4   , 8.0E4   , 9.0E4    ...
                   , 1186.5  , 1077.4  , 1036.4  , 994.8   , 968.1    ...
                   , 968.1   , 968.1   , 970.9   , 977.6   , 984.3   /
      "----------------------LOG OF ATMOSPHERIC DENSITY "
      TABLE          LRO, 1, 10   ...
                   / 0.0     , 1.0E4   , 2.0E4   , 3.0E4   , 4.0E4    ...
                   , 5.0E4   , 6.0E4   , 7.0E4   , 8.0E4   , 9.0E4    ...
                   ,-6.04191 ,-6.34502 ,-6.67084 ,-7.02346 ,-7.43995  ...
                   ,-7.91851 ,-8.39664 ,-8.87953 ,-9.36448 ,-9.87239/

      "----------------------MISSILE AIRFRAME MODULE "

      "----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      REAL           ME(9), VMM(3), NM(3), NME(3), DL(4), CD(3), C(6)
      REAL           VM(3), VMD(3), VMIC(3), RM(3), RMD(3), RMIC(3)
      REAL           WM(3), WMD(3), WMIC(3), A(30)

      "----------------------MISSILE DIMENSIONAL CONSTANTS "
      CONSTANT       B = 3.95            , CBAR = 5.62
      CONSTANT       S = 13.9            , DXREF = 9.60
      CONSTANT       DL = 4*0.0
      "----------------------INITIAL CONDITION VALUES "
      CONSTANT       SIMIC = 0.0         , THMIC = 0.0
      CONSTANT       FIMIC = 0.0         , WMIC  = 3*0.0
      CONSTANT       VMIC = 2154.8, 2*0.0
      CONSTANT       RMIC = 0.0, 10000.0, 0.0
      "----------------------DEFINE ELEMENTS OF STABILITY DERIVATIVE "
      " MATRIX. LINEAR AERODATA IS ASSUMED FOR SIMPLICITY IN SUBROUTINE"
      " COEFF. NON-LINEAR AERODATA MAY BE INCORPORATED BY REWRITING "
      " THIS SUBROUTINE "
      CONSTANT       A =      ...
           0.148  , 0.0    , 0.0    , 0.0    , 0.0    , 0.0     ...
         , 0.0    ,-0.26   , 0.0    , 0.0    , 0.0    ,-0.286   ...
         , 0.0    , 0.0    ,-0.26   , 0.0    , 0.286  , 0.0     ...
         , 0.0    , 0.528  , 0.0    , 0.0    , 0.0    , 2.0     ...
```

```
    , 0.0      , 0.0      , 0.528    , 0.0      ,-2.0      , 0.0
"-----------------------ROLL DAMPING - FUNCTION OF MACH NUMBER "
TABLE           CLP, 1, 5     ...
                / 0.0      , 0.8      , 1.0      , 1.2      , 2.0      ...
                ,-0.21     ,-0.21     ,-0.20     ,-0.19     ,-0.18    /
"-----------------------PITCH DAMPING - FUNCTION OF MACH NUMBER "
TABLE           CMQ, 1, 5     ...
                / 0.0      , 0.8      , 1.0      , 1.2      , 2.0      ...
                ,-3.8      ,-2.0      ,-1.5      ,-2.0      ,-2.1     /


"block 0 : motor, aerodynamics, and rotational velocity dynamics"
     procedural(WMD = CD, C, Q, WM)
      ZBLOCK=0

      "-----------------------MOTOR MODULE "

      "-----------------------SIMPLE VERSION WITH ZERO THRUST SPECIF- "
      "                       YING A BURNT OR GLIDE CONDITION "
      CONSTANT        THRUST = 0.0       , MASS = 8.77
      CONSTANT        IXX = 8.77         , IYY = 361.8
      CONSTANT        DXCG = 10.2


      "-----------------------CALCULATE ACCELERATION DUE TO AERODYNAMIC"
      "                       EFFECTS AND ROTATION RATE DERIVATIVES "
      WMD(1)   = Q*S*B*(C(1) + CD(1))/IXX
      WMD(2)   = Q*S*CBAR*(C(2) + CD(2))/IYY + WM(1)*WM(3)
      WMD(3)   = Q*S*CBAR*(C(3) + CD(3))/IYY - WM(1)*WM(2)
     end $"of procedural"

      "-----------------------VECTOR INTEGRATE FOR ROTATIONAL VELOCITY "
      WM       = INTVC(WMD, WMIC)
"end of block 0"


"block 1 : rotational posn dynamics"
     procedural(SIMD = WM, THM, FIM )
      ZBLOCK=1
      "-----------------------YAW ANGLE DERIVATIVE "
      SIMD     = (WM(2)*COS(FIM) - WM(3)*SIN(FIM))/COS(THM)

     end $"of procedural"

      "-----------------------INTEGRATE FOR ALL EULER ANGLES - NOTE USE"
      "                       OF VECTOR INTEGRATOR FOR SINGLE ELEMENT "
      SIM      = INTVC(SIMD, SIMIC)
      THM      = INTEG(WM(2)*SIN(FIM) + WM(3)*COS(FIM), THMIC)
      FIM      = INTEG(WM(1) - SIMD*SIN(THM), FIMIC)
"end of block 1"


"block 2 : translational velocity dynamics"
     procedural(VMD,Q,CD,C,ME = FIM,SIM,THM,RM,WM)
      ZBLOCK=2
      "-----------------------MAKE *ME* MATRIX FROM ORIENTATION ANGLES "
      CALL MMK(ME = FIM, 1, THM, 3, SIM, 2)
      "-----------------------CALCULATE ACTUAL ATMOSPHERIC DENSITY "
      RO       = EXP(LRO(RM(2)))

      "-----------------------MAGNITUDE OF MISSILE VELOCITY "
      MVM      = SQRT(DOT(VM, VM))
      "-----------------------ROTATE VELOCITY TO MISSILE FRAME "
      CALL VECROT(VMM = VM, ME)
      "-----------------------LATERAL AND VERTICAL ANGLES OF ATTACK "
      AL2      = ATAN(-VMM(3)/VMM(1))
      AL3      = ATAN( VMM(2)/VMM(1))
      "-----------------------MACH NUMBER AND DYNAMIC PRESSURE "
      MACH     = MVM/VS(RM(2))
      Q        = 0.5*RO*MVM**2
      "-----------------------CALCULATE DAMPING DERIVATIVES "
      CD(1)    = 0.5*CLP(MACH)*B*WM(1)/MVM
      CCVV     = 0.5*CMQ(MACH)*CBAR/MVM
      CD(2)    = CCVV*WM(2)
      CD(3)    = CCVV*WM(3)
      "-----------------------GET MOMENTS AND FORCE AERO COEFFICIENTS "
      " AND CORRECT LATERAL MOMENTS FOR SHIFT IN CENTRE OF GRAVITY "
      " POSITION "
      CALL COEFF(C = AL2, AL3, DL, MACH)
      C(2)     = C(2) - (DXCG - DXREF)*C(6)/CBAR
      C(3)     = C(3) + (DXCG - DXREF)*C(5)/CBAR
      "----next three lines have been moved from posn in missil2"
      NM(1)    = (Q*S*C(4) + THRUST)/MASS
```

```
      NM(2)    = Q*S*C(5)/MASS
      NM(3)    = Q*S*C(6)/MASS
      "-----------------------ROTATE ACCELERATION VECTOR TO EARTH FRAME"
      CALL INVROT(NME = NM, ME)
      "-----------------------CALCULATE VELOCITY DERIVATIVES IN THE  "
      "                       EARTH FRAME - NEEDS GRAVITY ADDING IN "
      VMD(1)   = NME(1)
      VMD(2)   = NME(2) - G
      VMD(3)   = NME(3)
     end $" of procedual "
      "-----------------------TRANSLATIONAL VELOCITY "
      VM       = INTVC(VMD, VMIC)
"end of block 2"


"block 3 : translational position dynamics"
     procedural(RMD = VM)
      ZBLOCK=3
      "-----------------------TRANSLATIONAL POSITION - NOTE THE DERIV- "
      " ATIVE VECTOR CANNOT BE A STATE VECTOR (VELOCITY) AS WELL "
      CALL XFERB(RMD = VM, 3)
     end $" of procedural "
      RM       = INTVC(RMD, RMIC)
"end of block 3"


END $" OF DERIVATIVE "

      "-----------------------STOP ON ELAPSED TIME "
      CONSTANT        TSTP = 1.99
      TERMT(T .GE. TSTP)

END $" OF DYNAMIC "

END $" OF PROGRAM "
      SUBROUTINE INIT(C)
C-----------------------------FORTRAN SUBROUTINE WHOSE ONLY JOB IS TO
C     TRANSFER THE STABILITY DERIVATIVE MATRIX TO AN ARRAY IN LABELLED
C     COMMON SO THAT IT MAY BE ACCESSED IN SUBROUTINE COEFF. NOTE NO
C     COMMON BLOCKS MAY BE DEFINED IN THE ACSL MODEL DEFINITION SECTION
C
      COMMON/STABD/  A(6,5)
      DATA           LENGTH  / 30     /
C
C-----------------------------TRANSFER BLOCK
      CALL XFERB(C, LENGTH, A)
      RETURN
C
      END
      SUBROUTINE COEFF(AL2, AL3, DL, MACH, C)
C-----------------------------COMPUTES SIX AERODYNAMIC COEFFICIENTS -
C     THREE MOMENTS, C(1), C(2) AND C(3), AND THREE FORCES, C(4), C(5)
C     AND C(6). MOMENTS ARE ABOUT AXES CENTRED AT THE REFERENCE POINT
C     AND MUST BE CORRECTED FOR CENTRE OF GAVITY SHIFT.
C
C     INPUTS
C
C     AL2      ANGLE OF ATTACK ABOUT *M2* - POSITIVE WIND FROM LEFT
C     AL3      ANGLE OF ATTACK ABOUT *M3* - POSITIVE WIND FROM ABOVE
C     DL       ARRAY OF FOUR FIN DEFLECTIONS
C     MACH     MACH NUMBER (REAL)
C
C     OUTPUTS
C
C     C        ARRAY OF SIX AERODYNAMIC COEFFICIENTS
C
      REAL           DL(4)   , C(6)    , MACH
C
      COMMON/STABD/  A(6,5)
C
C-----------------------------COMPUTE EQUIVALENT CONTROL SURFACE DEFL-
C                             ECTIONS FROM THE FOUR SURFACE ANGLES
      DLA      = 0.25*(DL(3) + DL(4) - DL(1) - DL(2))
      DLY      = 0.50*(DL(1) + DL(3))
      DLZ      = 0.50*(DL(2) + DL(4))
C-----------------------------COMPUTE EACH MOMENT ASSUMING IT IS LINEAR
C                             IN EACH OF THE ARGUMENTS
      DO 110 J = 1, 6
      C(J)     = A(J,1)*DLA + A(J,2)*DLY + A(J,3)*DLZ + A(J,4)*AL2
     +                      + A(J,5)*AL3
  110 CONTINUE
      RETURN
C
      END
```

```
      FUNCTION DOT (A, B)
C-----------------------------COMPUTE VECTOR DOT PRODUCT OF TWO VECTORS.
C
C     PROGRAMMER V. B. WAYLAND
C
C     INPUTS
C
C     A AND B ARE ARRAYS OF LENGTH 3
C
C     OUTPUT
C
C     DOT IS A REAL FUNCTION RETURNING THE DOT PRODUCT OF A  AND B
C
      DIMENSION      A(3)    , B(3)
C
      DOT      = A(1) * B(1)  +  A(2) * B(2)  +  A(3) * B(3)
      RETURN
C
      END
      SUBROUTINE INV ROT(VIN, RMX, VOUT)
C-----------------------------INVERSE VECTOR ROTATION. INVERSELY ROTATE
C     AN INPUT VECTOR, VIN, FROM ONE COORDINATE SYSTEM THRU A TRANSPOSED
C     ROTATION MATRIX, RMX. THE NEW VECTOR IS VOUT.
C
C     B = (AB)T * A
C
C     INPUT
C
C     VIN        INPUT 3 VECTOR
C     RMX        3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT       OUTPUT 3 VECTOR
C
      DIMENSION       VIN(3)   , RMX(3,3), VOUT(3)
C
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(2,1)*VIN(2) + RMX(3,1)*VIN(3)
      VOUT(2) = RMX(1,2)*VIN(1) + RMX(2,2)*VIN(2) + RMX(3,2)*VIN(3)
      VOUT(3) = RMX(1,3)*VIN(1) + RMX(2,3)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
      SUBROUTINE  MMK(A, NA, B, NB, C, NC, RM)
C     1A-0      30 DEC 68        MAKE A DIRECTION COSINE MATRIX
C-----------------------------ROUTINE GENERATES A DIRECTION COSINE MATRX
C     BY ROTATING IN ORDER
C
C     1)ANGLE C ABOUT THE NC AXIS
C     2)ANGLE B ABOUT THE NB AXIS
C     3)ANGLE A ABOUT THE NA AXIS
C
C     INPUTS
C
C     ANGLES A, B, C IN RADIANS
C     NA, NB, NC - A NUMBER BETWEEN 1 AND 3 CORRESPONDING TO AXIS
C                  ABOUT WHICH EACH ANGLE IS ROTATED
C
C     OUTPUT
C
C     RM -- A 3X3 DIRECTION COSINE MATRIX
C
      REAL           AM(3,3) , BM(3,3) , CM(3,3) , RM(3,3)
      REAL           T(9)
C
C     NOTE FOR FORMING A DIRECTION COSINE MATRIX FROM EULER ANGLES THE
C     CONVENTION IS TO ROTATE ANGLE PHI   ABOUT THE NO. 1 AXIS, ANGLE
C     PSI ABOUT THE NO. 2 AXIS AND ANGLE THETA ABOUT THE NO. 3 AXIS
C
C-----------------------------GENERATE THE ROTATION MATRIX FOR EACH ANG.
      CALL ROTMX ( A, NA, AM)
      CALL ROTMX ( B, NB, BM)
      CALL ROTMX ( C, NC, CM)
C-----------------------------MATRIX MULTIPLY THE INTERMEDIATE MATRICES
      CALL  MML XY(BM,CM,T)
      CALL  MML XY(AM,T,RM)
      RETURN
C
      END
      SUBROUTINE  MML XY (X, Y, Z)
C-----------------------------MATRIX MULTIPLY ROUTINES FOR TWO 3X3
C     MATRICES. FIRST ENTRY CONTAINS NO TRANSPOSES
C
C     Z = (X) * (Y)
C
```

```
C     INPUT
C
C     X          FIRST 3X3 MATRIX
C     Y          SECOND 3X3 MATRIX
C
C     OUTPUT
C
C     Z          RESULTING 3X3 MATRIX WHERE
C     Z(I,J)   = X(I,1)*Y(1,J) + X(I,2)*Y(2,J) + X(I,3)*Y(3,J)
C
      DIMENSION      X(3,3)   ,  Y(3,3) , Z(3,3)
C
      Z(1,1)   = X(1,1)*Y(1,1) + X(1,2)*Y(2,1) + X(1,3)*Y(3,1)
      Z(2,1)   = X(2,1)*Y(1,1) + X(2,2)*Y(2,1) + X(2,3)*Y(3,1)
      Z(3,1)   = X(3,1)*Y(1,1) + X(3,2)*Y(2,1) + X(3,3)*Y(3,1)
      Z(1,2)   = X(1,1)*Y(1,2) + X(1,2)*Y(2,2) + X(1,3)*Y(3,2)
      Z(2,2)   = X(2,1)*Y(1,2) + X(2,2)*Y(2,2) + X(2,3)*Y(3,2)
      Z(3,2)   = X(3,1)*Y(1,2) + X(3,2)*Y(2,2) + X(3,3)*Y(3,2)
      Z(1,3)   = X(1,1)*Y(1,3) + X(1,2)*Y(2,3) + X(1,3)*Y(3,3)
      Z(2,3)   = X(2,1)*Y(1,3) + X(2,2)*Y(2,3) + X(2,3)*Y(3,3)
      Z(3,3)   = X(3,1)*Y(1,3) + X(3,2)*Y(2,3) + X(3,3)*Y(3,3)
      RETURN
C
      END
      SUBROUTINE  ROTMX( X, I, XM)
C     1A-0     30 DEC 68        ROTATION MATRIX
C-----------------------------GENERATE BY STARTING WITH AN IDENTITY MATX
C     PUT THE COSINE OF ANGLE X ON THE DIAGONAL AND +SIN(X) AND -SIN(X)
C     ON OFF DIAGONALS
C
      REAL            XM(3,3)
      INTEGER         II T(3), III T(3)
C
      DATA     II T / 2 , 3  , 1  /
     .        ,III T/ 3 , 1  , 2  /
C
      SX       = SIN(X)
      CX       = COS(X)
      II       = II T(I)
      III      = III T(I)
C
      XM(I,I) = 1.0
      XM(I,II)= 0.0
      XM(II,I)= 0.0
      XM(I,III)= 0.0
      XM(III,I)= 0.0
C
      XM(II,II)= CX
      XM(III,III)= CX
      XM(II,III) =  SX
      XM(III,II) = -SX
C
      RETURN
C
      END
      SUBROUTINE  VEC ROT (VIN, RMX, VOUT)
C     1A-0   25 NOV 68          VECTOR ROTATION
C-----------------------------ROTATE AN INPUT VECTOR, VIN, FROM ONE
C     COORDINATE SYSTEM THRU A ROTATION MATRIX, RMX.  THE NEW VECTOR IS
C     VOUT.
C
C     A = (AB) * B
C
C     INPUT
C
C     VIN       INPUT 3 VECTOR
C     RMX       3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT      OUTPUT 3 VECTOR
C
      DIMENSION       VIN(3)   , RMX(3,3), VOUT(3)
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(1,2)*VIN(2) + RMX(1,3)*VIN(3)
      VOUT(2) = RMX(2,1)*VIN(1) + RMX(2,2)*VIN(2) + RMX(2,3)*VIN(3)
      VOUT(3) = RMX(3,1)*VIN(1) + RMX(3,2)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
```

APPENDIX D: MISSIL4.CSL

```
PROGRAM - MISSILE AIRFRAME MODEL

      " Modified from missil3.csl."
      " Removed (in block 0) the dependency on nonstate variables "
      " CD, C, and Q.  This was accomplished by replicating the "
      " code that evaluates these variables, so the corresponding "
      " lines appear in both block 0 and in block 2.  While this "
      " may seem wasteful, note that in a parallel implementation, "
      " the lines would be evaluated simultaneously.  If they were "
      " not replicated (as in missil3.csl), then the parallel im-"
      " plementation would have had to evaluate the blocks in "
      " stages, with some idle processors in each stage.  (There is "
      " another way of doing this, involving extra evaluations of "
      " the code on each processor, which is simpler to implement, "
      " but has the same effect.)  As an exercise, note that it is "
      " not absolutely necessary to replicate all of the CD/C/Q lines "
      " in both blocks, but the time savings is not very great. "

      " This model runs under ACSL and generates the same output as "
      " the original missile.csl model."

      "-----------------------A GENERIC MISSILE AIRFRAME MODEL IS "
      " DEVELOPED USING VECTORS FOR ALL THREE DIMENSIONAL QUANTITIES. "
      " THIS MODEL WILL RESPOND TO FIN DEFLECTIONS SO REPRESENTING THE "
      " OPEN LOOP AIRFRAME RESPONSE AND NEEDS A SEEKER, AUTOPILOT, "
      " ACTUATOR, MOTOR AND TARGET MODULE IN ORDER TO EVALUATE GUIDANCE"
      " EFFECTIVENESS "

INITIAL

      ALGORITHM      IALG = 4
      MAXTERVAL      MAXT = 0.010
      NSTEPS         NSTP = 1
      CINTERVAL      CINT = 0.020

      "-----------------------SET UP IN CASE DICTIONARY REQUIRED "
      LOGICAL        DICTDM            $ CONSTANT DICTDM = .FALSE.
      IF(DICTDM) CALL LISTD(5)
      DICTDM  = .FALSE.
      "-----------------------PASS STABILITY DERIVATIVE MATRIX TO THE "
      "                       COEFFICIENT GENERATION SUBROUTINE "
      CALL INIT(A)

END $" OF INITIAL "

DYNAMIC

DERIVATIVE

      "-----------------------ENVIRONMENT MODULE "

      "-----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      CONSTANT       G = 32.2
      "-----------------------VELOCITY OF SOUND - FUNCTION OF ALTITUDE "
      TABLE          VS, 1, 10    ...
                   / 0.0     , 1.0E4   , 2.0E4   , 3.0E4   , 4.0E4    ...
                   , 5.0E4   , 6.0E4   , 7.0E4   , 8.0E4   , 9.0E4    ...
                   , 1186.5  , 1077.4  , 1036.4  , 994.8   , 968.1    ...
                   , 968.1   , 968.1   , 970.9   , 977.6   , 984.3   /
      "-----------------------LOG OF ATMOSPHERIC DENSITY "
      TABLE          LRO, 1, 10   ...
                   / 0.0     , 1.0E4   , 2.0E4   , 3.0E4   , 4.0E4    ...
                   , 5.0E4   , 6.0E4   , 7.0E4   , 8.0E4   , 9.0E4    ...
                   ,-6.04191 ,-6.34502 ,-6.67084 ,-7.02346 ,-7.43995 ...
                   ,-7.91851 ,-8.39664 ,-8.87953 ,-9.36448 ,-9.87239/

      "-----------------------MISSILE AIRFRAME MODULE "

      "-----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      REAL           ME(9), VMM(3), NM(3), NME(3), DL(4), CD(3), C(6)
      REAL           VM(3), VMD(3), VMIC(3), RM(3), RMD(3), RMIC(3)
      REAL           WM(3), WMD(3), WMIC(3), A(30)

      "-----------------------MISSILE DIMENSIONAL CONSTANTS "
      CONSTANT       B = 3.95          , CBAR = 5.62
      CONSTANT       S = 13.9          , DXREF = 9.60
      CONSTANT       DL = 4*0.0
      "-----------------------INITIAL CONDITION VALUES "
      CONSTANT       SIMIC = 0.0       , THMIC = 0.0
      CONSTANT       FIMIC = 0.0       , WMIC  = 3*0.0
      CONSTANT       VMIC = 2154.8, 2*0.0
      CONSTANT       RMIC = 0.0, 10000.0, 0.0
```

```
"-----------------------DEFINE ELEMENTS OF STABILITY DERIVATIVE "
" MATRIX. LINEAR AERODATA IS ASSUMED FOR SIMPLICITY IN SUBROUTINE"
" COEFF. NON-LINEAR AERODATA MAY BE INCORPORATED BY REWRITING "
" THIS SUBROUTINE "
CONSTANT        A =         ...
         0.148  , 0.0       , 0.0      , 0.0      , 0.0      , 0.0      ...
       , 0.0    ,-0.26      , 0.0      , 0.0      , 0.0      ,-0.286    ...
       , 0.0    , 0.0       ,-0.26     , 0.0      , 0.286    , 0.0      ...
       , 0.0    , 0.528     , 0.0      , 0.0      , 0.0      , 2.0      ...
       , 0.0    , 0.0       , 0.528    , 0.0      ,-2.0      , 0.0
"-----------------------ROLL DAMPING - FUNCTION OF MACH NUMBER "
TABLE           CLP, 1, 5    ...
               / 0.0       , 0.8      , 1.0      , 1.2      , 2.0      ...
               ,-0.21      ,-0.21     ,-0.20     ,-0.19     ,-0.18     /
"-----------------------PITCH DAMPING - FUNCTION OF MACH NUMBER "
TABLE           CMQ, 1, 5    ...
               / 0.0       , 0.8      , 1.0      , 1.2      , 2.0      ...
               ,-3.8       ,-2.0      ,-1.5      ,-2.0      ,-2.1      /


"block 0 : motor, aerodynamics, and rotational velocity dynamics"
    procedural(WMD = VM, FIM, THM, SIM, WM)
     ZBLOCK=0
     "-----------------------MAKE *ME* MATRIX FROM ORIENTATION ANGLES "
     CALL MMK(ME = FIM, 1, THM, 3, SIM, 2)
     "-----------------------CALCULATE ACTUAL ATMOSPHERIC DENSITY "
     RO      = EXP(LRO(RM(2)))

     "-----------------------MAGNITUDE OF MISSILE VELOCITY "
     MVM     = SQRT(DOT(VM, VM))
     "-----------------------ROTATE VELOCITY TO MISSILE FRAME "
     CALL VECROT(VMM = VM, ME)
     "-----------------------LATERAL AND VERTICAL ANGLES OF ATTACK "
     AL2     = ATAN(-VMM(3)/VMM(1))
     AL3     = ATAN( VMM(2)/VMM(1))
     "-----------------------MACH NUMBER AND DYNAMIC PRESSURE "
     MACH    = MVM/VS(RM(2))
     Q       = 0.5*RO*MVM**2
     "-----------------------CALCULATE DAMPING DERIVATIVES "
     CD(1)   = 0.5*CLP(MACH)*B*WM(1)/MVM
     CCVV    = 0.5*CMQ(MACH)*CBAR/MVM
     CD(2)   = CCVV*WM(2)
     CD(3)   = CCVV*WM(3)
     "-----------------------GET MOMENTS AND FORCE AERO COEFFICIENTS "
     " AND CORRECT LATERAL MOMENTS FOR SHIFT IN CENTRE OF GRAVITY "
     " POSITION "
     CALL COEFF(C = AL2, AL3, DL, MACH)
     C(2)    = C(2) - (DXCG - DXREF)*C(6)/CBAR
     C(3)    = C(3) + (DXCG - DXREF)*C(5)/CBAR

     "-----------------------MOTOR MODULE "

     "-----------------------SIMPLE VERSION WITH ZERO THRUST SPECIF- "
     "                       YING A BURNT OR GLIDE CONDITION "
     CONSTANT        THRUST = 0.0       , MASS = 8.77
     CONSTANT        IXX = 8.77         , IYY = 361.8
     CONSTANT        DXCG = 10.2


     "-----------------------CALCULATE ACCELERATION DUE TO AERODYNAMIC"
     "                       EFFECTS AND ROTATION RATE DERIVATIVES "
     WMD(1)  = Q*S*B*(C(1) + CD(1))/IXX
     WMD(2)  = Q*S*CBAR*(C(2) + CD(2))/IYY + WM(1)*WM(3)
     WMD(3)  = Q*S*CBAR*(C(3) + CD(3))/IYY - WM(1)*WM(2)
    end $"of procedural"

     "-----------------------VECTOR INTEGRATE FOR ROTATIONAL VELOCITY "
     WM      = INTVC(WMD, WMIC)
"end of block 0"


"block 1 : rotational posn dynamics"
    procedural(SIMD = WM, THM, FIM )
     ZBLOCK=1
     "-----------------------YAW ANGLE DERIVATIVE "
     SIMD    = (WM(2)*COS(FIM) - WM(3)*SIN(FIM))/COS(THM)

    end $"of procedural"

     "-----------------------INTEGRATE FOR ALL EULER ANGLES - NOTE USE"
     "                       OF VECTOR INTEGRATOR FOR SINGLE ELEMENT "
     SIM     = INTVC(SIMD, SIMIC)
     THM     = INTEG(WM(2)*SIN(FIM) + WM(3)*COS(FIM), THMIC)
     FIM     = INTEG(WM(1) - SIMD*SIN(THM), FIMIC)
```

```
"end of block 1"


"block 2 : translational velocity dynamics"
     procedural(VMD = VM, FIM, THM,SIM,RM,WM)
      ZBLOCK=2
      "-----------------------MAKE *ME* MATRIX FROM ORIENTATION ANGLES "
      CALL MMK(ME = FIM, 1, THM, 3, SIM, 2)
      "-----------------------CALCULATE ACTUAL ATMOSPHERIC DENSITY "
      RO      = EXP(LRO(RM(2)))

      "-----------------------MAGNITUDE OF MISSILE VELOCITY "
      MVM     = SQRT(DOT(VM, VM))
      "-----------------------ROTATE VELOCITY TO MISSILE FRAME "
      CALL VECROT(VMM = VM, ME)
      "-----------------------LATERAL AND VERTICAL ANGLES OF ATTACK "
      AL2     = ATAN(-VMM(3)/VMM(1))
      AL3     = ATAN( VMM(2)/VMM(1))
      "-----------------------MACH NUMBER AND DYNAMIC PRESSURE "
      MACH    = MVM/VS(RM(2))
      Q       = 0.5*RO*MVM**2
      "-----------------------CALCULATE DAMPING DERIVATIVES "
      CD(1)   = 0.5*CLP(MACH)*B*WM(1)/MVM
      CCVV    = 0.5*CMQ(MACH)*CBAR/MVM
      CD(2)   = CCVV*WM(2)
      CD(3)   = CCVV*WM(3)
      "-----------------------GET MOMENTS AND FORCE AERO COEFFICIENTS "
      " AND CORRECT LATERAL MOMENTS FOR SHIFT IN CENTRE OF GRAVITY "
      " POSITION "
      CALL COEFF(C = AL2, AL3, DL, MACH)
      C(2)    = C(2) - (DXCG - DXREF)*C(6)/CBAR
      C(3)    = C(3) + (DXCG - DXREF)*C(5)/CBAR
      "----next three lines have been moved from posn in missil2"
      NM(1)   = (Q*S*C(4) + THRUST)/MASS
      NM(2)   = Q*S*C(5)/MASS
      NM(3)   = Q*S*C(6)/MASS
      "-----------------------ROTATE ACCELERATION VECTOR TO EARTH FRAME"
      CALL INVROT(NME = NM, ME)
      "-----------------------CALCULATE VELOCITY DERIVATIVES IN THE  "
      "                       EARTH FRAME - NEEDS GRAVITY ADDING IN "
      VMD(1)  = NME(1)
      VMD(2)  = NME(2) - G
      VMD(3)  = NME(3)
     end $" of procedual "
      "-----------------------TRANSLATIONAL VELOCITY "
      VM      = INTVC(VMD, VMIC)
"end of block 2"


"block 3 : translational position dynamics"
     procedural(RMD = VM)
      ZBLOCK=3
      "-----------------------TRANSLATIONAL POSITION - NOTE THE DERIV- "
      " ATIVE VECTOR CANNOT BE A STATE VECTOR (VELOCITY) AS WELL "
      CALL XFERB(RMD = VM, 3)
     end $" of procedural "
      RM      = INTVC(RMD, RMIC)
"end of block 3"


END $" OF DERIVATIVE "

      "-----------------------STOP ON ELAPSED TIME "
      CONSTANT       TSTP = 1.99
      TERMT(T .GE. TSTP)

END $" OF DYNAMIC "

END $" OF PROGRAM "
      SUBROUTINE INIT(C)
C----------------------------FORTRAN SUBROUTINE WHOSE ONLY JOB IS TO
C     TRANSFER THE STABILITY DERIVATIVE MATRIX TO AN ARRAY IN LABELLED
C     COMMON SO THAT IT MAY BE ACCESSED IN SUBROUTINE COEFF. NOTE NO
C     COMMON BLOCKS MAY BE DEFINED IN THE ACSL MODEL DEFINITION SECTION
C
      COMMON/STABD/  A(6,5)
      DATA           LENGTH  / 30     /
C
C----------------------------TRANSFER BLOCK
      CALL XFERB(C, LENGTH, A)
      RETURN
C
      END
```

```
      SUBROUTINE COEFF(AL2, AL3, DL, MACH, C)
C-----------------------------COMPUTES SIX AERODYNAMIC COEFFICIENTS -
C     THREE MOMENTS, C(1), C(2) AND C(3), AND THREE FORCES, C(4), C(5)
C     AND C(6). MOMENTS ARE ABOUT AXES CENTRED AT THE REFERENCE POINT
C     AND MUST BE CORRECTED FOR CENTRE OF GAVITY SHIFT.
C
C     INPUTS
C
C     AL2      ANGLE OF ATTACK ABOUT *M2* - POSITIVE WIND FROM LEFT
C     AL3      ANGLE OF ATTACK ABOUT *M3* - POSITIVE WIND FROM ABOVE
C     DL       ARRAY OF FOUR FIN DEFLECTIONS
C     MACH     MACH NUMBER (REAL)
C
C     OUTPUTS
C
C     C        ARRAY OF SIX AERODYNAMIC COEFFICIENTS
C
      REAL            DL(4)   , C(6)    , MACH
C
      COMMON/STABD/  A(6,5)
C
C-----------------------------COMPUTE EQUIVALENT CONTROL SURFACE DEFL-
C                             ECTIONS FROM THE FOUR SURFACE ANGLES
      DLA      = 0.25*(DL(3) + DL(4) - DL(1) - DL(2))
      DLY      = 0.50*(DL(1) + DL(3))
      DLZ      = 0.50*(DL(2) + DL(4))
C-----------------------------COMPUTE EACH MOMENT ASSUMING IT IS LINEAR
C                             IN EACH OF THE ARGUMENTS
      DO 110 J = 1, 6
      C(J)     = A(J,1)*DLA + A(J,2)*DLY + A(J,3)*DLZ + A(J,4)*AL2
     .                    + A(J,5)*AL3
  110 CONTINUE
      RETURN
C
      END
      FUNCTION DOT (A, B)
C-----------------------------COMPUTE VECTOR DOT PRODUCT OF TWO VECTORS.
C
C     PROGRAMMER V. B. WAYLAND
C
C     INPUTS
C
C     A AND B ARE ARRAYS OF LENGTH 3
C
C     OUTPUT
C
C     DOT IS A REAL FUNCTION RETURNING THE DOT PRODUCT OF A  AND B
C
      DIMENSION       A(3)    , B(3)
C
      DOT      = A(1) * B(1)  +  A(2) * B(2)  +  A(3) * B(3)
      RETURN
C
      END
      SUBROUTINE INV ROT(VIN, RMX, VOUT)
C-----------------------------INVERSE VECTOR ROTATION. INVERSELY ROTATE
C     AN INPUT VECTOR, VIN, FROM ONE COORDINATE SYSTEM THRU A TRANSPOSED
C     ROTATION MATRIX, RMX. THE NEW VECTOR IS VOUT.
C
C     B = (AB)T * A
C
C     INPUT
C
C     VIN        INPUT 3 VECTOR
C     RMX        3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT       OUTPUT 3 VECTOR
C
      DIMENSION       VIN(3)  , RMX(3,3), VOUT(3)
C
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(2,1)*VIN(2) + RMX(3,1)*VIN(3)
      VOUT(2) = RMX(1,2)*VIN(1) + RMX(2,2)*VIN(2) + RMX(3,2)*VIN(3)
      VOUT(3) = RMX(1,3)*VIN(1) + RMX(2,3)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
      SUBROUTINE  MMK(A, NA, B, NB, C, NC, RM)
C     1A-0     30 DEC 68       MAKE A DIRECTION COSINE MATRIX
C-----------------------------ROUTINE GENERATES A DIRECTION COSINE MATRX
C     BY ROTATING IN ORDER
C
C     1)ANGLE C ABOUT THE NC AXIS
C     2)ANGLE B ABOUT THE NB AXIS
```

```
C     3)ANGLE A ABOUT THE NA AXIS
C
C     INPUTS
C
C     ANGLES A, B, C IN RADIANS
C     NA, NB, NC - A NUMBER BETWEEN 1 AND 3 CORRESPONDING TO AXIS
C                  ABOUT WHICH EACH ANGLE IS ROTATED
C
C     OUTPUT
C
C     RM -- A 3X3 DIRECTION COSINE MATRIX
C
      REAL          AM(3,3) , BM(3,3) , CM(3,3) , RM(3,3)
      REAL          T(9)
C
C     NOTE FOR FORMING A DIRECTION COSINE MATRIX FROM EULER ANGLES THE
C     CONVENTION IS TO ROTATE ANGLE PHI   ABOUT THE NO. 1 AXIS, ANGLE
C     PSI ABOUT THE NO. 2 AXIS AND ANGLE THETA ABOUT THE NO. 3 AXIS
C
C-----------------------------GENERATE THE ROTATION MATRIX FOR EACH ANG.
      CALL ROTMX ( A, NA, AM)
      CALL ROTMX ( B, NB, BM)
      CALL ROTMX ( C, NC, CM)
C-----------------------------MATRIX MULTIPLY THE INTERMEDIATE MATRICES
      CALL  MML XY(BM,CM,T)
      CALL  MML XY(AM,T,RM)
      RETURN
C
      END
      SUBROUTINE  MML XY (X, Y, Z)
C-----------------------------MATRIX MULTIPLY ROUTINES FOR TWO 3X3
C     MATRICES. FIRST ENTRY CONTAINS NO TRANSPOSES
C
C     Z = (X) * (Y)
C
C     INPUT
C
C     X          FIRST 3X3 MATRIX
C     Y          SECOND 3X3 MATRIX
C
C     OUTPUT
C
C     Z          RESULTING 3X3 MATRIX WHERE
C     Z(I,J)  = X(I,1)*Y(1,J) + X(I,2)*Y(2,J) + X(I,3)*Y(3,J)
C
      DIMENSION      X(3,3)  ,  Y(3,3) , Z(3,3)
C
      Z(1,1)  = X(1,1)*Y(1,1) + X(1,2)*Y(2,1) + X(1,3)*Y(3,1)
      Z(2,1)  = X(2,1)*Y(1,1) + X(2,2)*Y(2,1) + X(2,3)*Y(3,1)
      Z(3,1)  = X(3,1)*Y(1,1) + X(3,2)*Y(2,1) + X(3,3)*Y(3,1)
      Z(1,2)  = X(1,1)*Y(1,2) + X(1,2)*Y(2,2) + X(1,3)*Y(3,2)
      Z(2,2)  = X(2,1)*Y(1,2) + X(2,2)*Y(2,2) + X(2,3)*Y(3,2)
      Z(3,2)  = X(3,1)*Y(1,2) + X(3,2)*Y(2,2) + X(3,3)*Y(3,2)
      Z(1,3)  = X(1,1)*Y(1,3) + X(1,2)*Y(2,3) + X(1,3)*Y(3,3)
      Z(2,3)  = X(2,1)*Y(1,3) + X(2,2)*Y(2,3) + X(2,3)*Y(3,3)
      Z(3,3)  = X(3,1)*Y(1,3) + X(3,2)*Y(2,3) + X(3,3)*Y(3,3)
      RETURN
C
      END
      SUBROUTINE  ROTMX( X, I, XM)
C     1A-0     30 DEC 68       ROTATION MATRIX
C-----------------------------GENERATE BY STARTING WITH AN IDENTITY MATX
C     PUT THE COSINE OF ANGLE X ON THE DIAGONAL AND +SIN(X) AND -SIN(X)
C     ON OFF DIAGONALS
C
      REAL          XM(3,3)
      INTEGER       II T(3), III T(3)
C
      DATA     II T / 2 , 3  , 1  /
     .        ,III T/ 3 , 1  , 2  /
C
      SX      = SIN(X)
      CX      = COS(X)
      II      = II T(I)
      III     = III T(I)
C
      XM(I,I) = 1.0
      XM(I,II)= 0.0
      XM(II,I)= 0.0
      XM(I,III)= 0.0
      XM(III,I)= 0.0
C
      XM(II,II)= CX
      XM(III,III)= CX
      XM(II,III) =  SX
      XM(III,II) = -SX
```

```
C
      RETURN
C
      END
      SUBROUTINE  VEC ROT (VIN, RMX, VOUT)
C     1A-0   25 NOV 68          VECTOR ROTATION
C----------------------------ROTATE AN INPUT VECTOR, VIN, FROM ONE
C     COORDINATE SYSTEM THRU A ROTATION MATRIX, RMX.  THE NEW VECTOR IS
C     VOUT.
C
C     A = (AB) * B
C
C     INPUT
C
C     VIN      INPUT 3 VECTOR
C     RMX      3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT     OUTPUT 3 VECTOR
C
      DIMENSION      VIN(3)   , RMX(3,3), VOUT(3)
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(1,2)*VIN(2) + RMX(1,3)*VIN(3)
      VOUT(2) = RMX(2,1)*VIN(1) + RMX(2,2)*VIN(2) + RMX(2,3)*VIN(3)
      VOUT(3) = RMX(3,1)*VIN(1) + RMX(3,2)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
```

APPENDIX E: MISSIL5.CSL

```
PROGRAM - MISSILE AIRFRAME MODEL

      " Modified from missil4.csl."
      " Makes an approximation by evaluating aerodynamics coeffs and "
      " atmospheric damping in parallel with all derivatives. This "
      " has the effect of declaring C, CD, and Q as state variables, "
      " and they are always used one integration time step after "
      " they are evaluated (i. e., they are stale by DT).  The result,"
      " however, is a significantly faster simulation (if run in "
      " parallel) with virtually identical results, because we care-"
      " fully selected slowly-changing subsystems to make into false "
      " state variables.  Note how many fewer lines are in the worst "
      " block, compared with missil3. "

      "----------------------A GENERIC MISSILE AIRFRAME MODEL IS "
      " DEVELOPED USING VECTORS FOR ALL THREE DIMENSIONAL QUANTITIES. "
      " THIS MODEL WILL RESPOND TO FIN DEFLECTIONS SO REPRESENTING THE "
      " OPEN LOOP AIRFRAME RESPONSE AND NEEDS A SEEKER, AUTOPILOT, "
      " ACTUATOR, MOTOR AND TARGET MODULE IN ORDER TO EVALUATE GUIDANCE"
      " EFFECTIVENESS "

      "----------------------ENVIRONMENT MODULE "

      "----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      CONSTANT        G = 32.2
      "----------------------VELOCITY OF SOUND - FUNCTION OF ALTITUDE "
      TABLE           VS, 1, 10    ...
                    / 0.0      , 1.0E4    , 2.0E4    , 3.0E4    , 4.0E4    ...
                    , 5.0E4    , 6.0E4    , 7.0E4    , 8.0E4    , 9.0E4    ...
                    , 1186.5   , 1077.4   , 1036.4   , 994.8    , 968.1    ...
                    , 968.1    , 968.1    , 970.9    , 977.6    , 984.3   /
      "----------------------LOG OF ATMOSPHERIC DENSITY "
      TABLE           LRO, 1, 10   ...
                    / 0.0      , 1.0E4    , 2.0E4    , 3.0E4    , 4.0E4    ...
                    , 5.0E4    , 6.0E4    , 7.0E4    , 8.0E4    , 9.0E4    ...
                    ,-6.04191  ,-6.34502  ,-6.67084  ,-7.02346  ,-7.43995  ...
                    ,-7.91851  ,-8.39664  ,-8.87953  ,-9.36448  ,-9.87239/

      "----------------------MISSILE AIRFRAME MODULE "

      "----------------------DEFINE ARRAYS AND CONSTANTS FOR MODULE "
      REAL            ME(9), VMM(3), NM(3), NME(3), DL(4), CD(3), C(6)
      REAL            VM(3), VMD(3), VMIC(3), RM(3), RMD(3), RMIC(3)
      REAL            WM(3), WMD(3), WMIC(3), A(30)
      real            CDIC(3), CDDOT(3), CIC(6), CDOT(6)

      "----------------------MISSILE DIMENSIONAL CONSTANTS "
      CONSTANT        B = 3.95             , CBAR = 5.62
      CONSTANT        S = 13.9             , DXREF = 9.60
      CONSTANT        DL = 4*0.0
      "----------------------INITIAL CONDITION VALUES "
      CONSTANT        SIMIC = 0.0          , THMIC = 0.0
      CONSTANT        FIMIC = 0.0          , WMIC  = 3*0.0
      CONSTANT        VMIC = 2154.8, 2*0.0
      CONSTANT        RMIC = 0.0, 10000.0, 0.0
      "----------------------DEFINE ELEMENTS OF STABILITY DERIVATIVE "
      " MATRIX. LINEAR AERODATA IS ASSUMED FOR SIMPLICITY IN SUBROUTINE"
      " COEFF. NON-LINEAR AERODATA MAY BE INCORPORATED BY REWRITING "
      " THIS SUBROUTINE "
      CONSTANT        A =          ...
          0.148     , 0.0      , 0.0      , 0.0      , 0.0      , 0.0      ...
        , 0.0       ,-0.26     , 0.0      , 0.0      , 0.0      ,-0.286    ...
        , 0.0       , 0.0      ,-0.26     , 0.0      , 0.286    , 0.0      ...
        , 0.0       , 0.528    , 0.0      , 0.0      , 0.0      , 2.0      ...
        , 0.0       , 0.0      , 0.528    , 0.0      ,-2.0      , 0.0
      "----------------------ROLL DAMPING - FUNCTION OF MACH NUMBER "
      TABLE           CLP, 1, 5    ...
                    / 0.0      , 0.8      , 1.0      , 1.2      , 2.0      ...
                    ,-0.21     ,-0.21     ,-0.20     ,-0.19     ,-0.18    /
      "----------------------PITCH DAMPING - FUNCTION OF MACH NUMBER "
      TABLE           CMQ, 1, 5    ...
                    / 0.0      , 0.8      , 1.0      , 1.2      , 2.0      ...
                    ,-3.8      ,-2.0      ,-1.5      ,-2.0      ,-2.1     /

INITIAL

      ALGORITHM       IALG = 4
      MAXTERVAL       MAXT = 0.010
      NSTEPS          NSTP = 1
      CINTERVAL       CINT = 0.020

      "----------------------SET UP IN CASE DICTIONARY REQUIRED "
```

```
      LOGICAL        DICTDM             $ CONSTANT DICTDM = .FALSE.
      IF(DICTDM) CALL LISTD(5)
      DICTDM  = .FALSE.
      "-----------------------PASS STABILITY DERIVATIVE MATRIX TO THE "
      "                       COEFFICIENT GENERATION SUBROUTINE "
      CALL INIT(A)

      "preevaluate C, CD, and Q initial conditions"
      CALL MMK(ME = FIMIC, 1, THMIC, 3, SIMIC, 2)
      "-----------------------CALCULATE ACTUAL ATMOSPHERIC DENSITY "
      RO      = EXP(LRO(RMIC(2)))

      "-----------------------MAGNITUDE OF MISSILE VELOCITY "
      MVM     = SQRT(DOT(VMIC, VMIC))
      "-----------------------ROTATE VELOCITY TO MISSILE FRAME "
      CALL VECROT(VMM = VMIC, ME)
      "-----------------------LATERAL AND VERTICAL ANGLES OF ATTACK "
      AL2     = ATAN(-VMM(3)/VMM(1))
      AL3     = ATAN( VMM(2)/VMM(1))
      "-----------------------MACH NUMBER AND DYNAMIC PRESSURE "
      MACH    = MVM/VS(RMIC(2))
      Q       = 0.5*RO*MVM**2
      "-----------------------CALCULATE DAMPING DERIVATIVES "
      CDIC(1)   = 0.5*CLP(MACH)*B*WMIC(1)/MVM
      CCVV      = 0.5*CMQ(MACH)*CBAR/MVM
      CDIC(2)   = CCVV*WMIC(2)
      CDIC(3)   = CCVV*WMIC(3)
      "-----------------------GET MOMENTS AND FORCE AERO COEFFICIENTS "
      " AND CORRECT LATERAL MOMENTS FOR SHIFT IN CENTRE OF GRAVITY "
      " POSITION "
      CALL COEFF(CIC = AL2, AL3, DL, MACH)
      CIC(2)    = CIC(2) - (DXCG - DXREF)*CIC(6)/CBAR
      CIC(3)    = CIC(3) + (DXCG - DXREF)*CIC(5)/CBAR

END $" OF INITIAL "

DYNAMIC

DERIVATIVE


"block 0 : aerodynamics, and rotational velocity dynamics"
     procedural(WMD = CD, Q, C, WM)
      ZBLOCK=0

      CONSTANT        IXX = 8.77         , IYY = 361.8


      "-----------------------CALCULATE ACCELERATION DUE TO AERODYNAMIC"
      "                       EFFECTS AND ROTATION RATE DERIVATIVES "
      WMD(1)  = Q*S*B*(C(1) + CD(1))/IXX
      WMD(2)  = Q*S*CBAR*(C(2) + CD(2))/IYY + WM(1)*WM(3)
      WMD(3)  = Q*S*CBAR*(C(3) + CD(3))/IYY - WM(1)*WM(2)
     end $"of procedural"

      "-----------------------VECTOR INTEGRATE FOR ROTATIONAL VELOCITY "
      WM      = INTVC(WMD, WMIC)
"end of block 0"


"block 1 : rotational posn dynamics"
     procedural(SIMD = WM, THM, FIM )
      ZBLOCK=1
      "-----------------------YAW ANGLE DERIVATIVE "
      SIMD    = (WM(2)*COS(FIM) - WM(3)*SIN(FIM))/COS(THM)

     end $"of procedural"

      "-----------------------INTEGRATE FOR ALL EULER ANGLES - NOTE USE"
      "                       OF VECTOR INTEGRATOR FOR SINGLE ELEMENT "
      SIM     = INTVC(SIMD, SIMIC)
      THM     = INTEG(WM(2)*SIN(FIM) + WM(3)*COS(FIM), THMIC)
      FIM     = INTEG(WM(1) - SIMD*SIN(THM), FIMIC)
"end of block 1"


"block 2 : translational velocity dynamics"
     procedural(VMD = Q, C, FIM, THM, SIM)
      ZBLOCK=2
      "-----------------------MOTOR MODULE "

      "-----------------------SIMPLE VERSION WITH ZERO THRUST SPECIF- "
      "                       YING A BURNT OR GLIDE CONDITION "
```

```
      CONSTANT       THRUST = 0.0       , MASS = 8.77
      "----------------------MAKE *ME* MATRIX FROM ORIENTATION ANGLES "
      CALL MMK(ME = FIM, 1, THM, 3, SIM, 2)
      "----next three lines have been moved from posn in missil2"
      NM(1)   = (Q*S*C(4) + THRUST)/MASS
      NM(2)   = Q*S*C(5)/MASS
      NM(3)   = Q*S*C(6)/MASS
      "----------------------ROTATE ACCELERATION VECTOR TO EARTH FRAME"
      CALL INVROT(NME = NM, ME)
      "----------------------CALCULATE VELOCITY DERIVATIVES IN THE  "
      "                      EARTH FRAME - NEEDS GRAVITY ADDING IN "
      VMD(1)  = NME(1)
      VMD(2)  = NME(2) - G
      VMD(3)  = NME(3)
     end $" of procedual "
      "----------------------TRANSLATIONAL VELOCITY "
      VM      = INTVC(VMD, VMIC)
"end of block 2"


"block 3 : translational position dynamics"
     procedural(RMD = VM)
      ZBLOCK=3
      "----------------------TRANSLATIONAL POSITION - NOTE THE DERIV- "
      " ATIVE VECTOR CANNOT BE A STATE VECTOR (VELOCITY) AS WELL "
      CALL XFERB(RMD = VM, 3)
     end $" of procedural "
      RM      = INTVC(RMD, RMIC)
"end of block 3"


"block 4 : atmospheric damping "
     procedural(CDDOT = WM, VM, RM)
      ZBLOCK=4
      "----------------------MAGNITUDE OF MISSILE VELOCITY "
      MVM     = SQRT(DOT(VM, VM))
      "----------------------CALCULATE ACTUAL ATMOSPHERIC DENSITY "
      RO      = EXP(LRO(RM(2)))
      "----------------------MACH NUMBER AND DYNAMIC PRESSURE "
      MACH    = MVM/VS(RM(2))
      Q       = 0.5*RO*MVM**2
      "----------------------CALCULATE DAMPING DERIVATIVES "
      CD(1)   = 0.5*CLP(MACH)*B*WM(1)/MVM
      CCVV    = 0.5*CMQ(MACH)*CBAR/MVM
      CD(2)   = CCVV*WM(2)
      CD(3)   = CCVV*WM(3)
      "make zero derivatives for false state"
      CDDOT(1)  = 0
      CDDOT(2)  = 0
      CDDOT(3)  = 0
     end $" of procedural "
      "perform an integration on our false states that will not "
      "change the values assigned above "
      CD      = intvc(CDDOT, CDIC)
"end of block 4"


"block 5 : aerodynamic coefficients "
     procedural(CDOT = FIM, THM, SIM, RM, VM)
      ZBLOCK=5
      CONSTANT       DXCG = 10.2
      "----------------------MAKE *ME* MATRIX FROM ORIENTATION ANGLES "
      CALL MMK(ME = FIM, 1, THM, 3, SIM, 2)
      "----------------------MAGNITUDE OF MISSILE VELOCITY "
      MVM     = SQRT(DOT(VM, VM))
      "----------------------ROTATE VELOCITY TO MISSILE FRAME "
      CALL VECROT(VMM = VM, ME)
      "----------------------LATERAL AND VERTICAL ANGLES OF ATTACK "
      AL2     = ATAN(-VMM(3)/VMM(1))
      AL3     = ATAN( VMM(2)/VMM(1))
      "----------------------MACH NUMBER but not DYNAMIC PRESSURE "
      MACH    = MVM/VS(RM(2))
      "----------------------GET MOMENTS AND FORCE AERO COEFFICIENTS "
      " AND CORRECT LATERAL MOMENTS FOR SHIFT IN CENTRE OF GRAVITY "
      " POSITION "
      CALL COEFF(C = AL2, AL3, DL, MACH)
      C(2)    = C(2) - (DXCG - DXREF)*C(6)/CBAR
      C(3)    = C(3) + (DXCG - DXREF)*C(5)/CBAR
      "make zero derivatives for false state"
      CDOT(1)  = 0
      CDOT(2)  = 0
      CDOT(3)  = 0
      CDOT(4)  = 0
```

```
      CDOT(5)  = 0
      CDOT(6)  = 0
     end $" of procedural "
      "perform an integration on our false states that will not "
      "change the values assigned above "
      C        = intvc(CDOT, CIC)
"end of block 5"


END $" OF DERIVATIVE "

      "-----------------------STOP ON ELAPSED TIME "
      CONSTANT       TSTP = 1.99
      TERMT(T .GE. TSTP)

END $" OF DYNAMIC "

END $" OF PROGRAM "
      SUBROUTINE INIT(C)
C-----------------------------FORTRAN SUBROUTINE WHOSE ONLY JOB IS TO
C     TRANSFER THE STABILITY DERIVATIVE MATRIX TO AN ARRAY IN LABELLED
C     COMMON SO THAT IT MAY BE ACCESSED IN SUBROUTINE COEFF. NOTE NO
C     COMMON BLOCKS MAY BE DEFINED IN THE ACSL MODEL DEFINITION SECTION
C
      COMMON/STABD/  A(6,5)
      DATA           LENGTH  / 30    /
C
C-----------------------------TRANSFER BLOCK
      CALL XFERB(C, LENGTH, A)
      RETURN
C
      END
      SUBROUTINE COEFF(AL2, AL3, DL, MACH, C)
C-----------------------------COMPUTES SIX AERODYNAMIC COEFFICIENTS -
C     THREE MOMENTS, C(1), C(2) AND C(3), AND THREE FORCES, C(4), C(5)
C     AND C(6). MOMENTS ARE ABOUT AXES CENTRED AT THE REFERENCE POINT
C     AND MUST BE CORRECTED FOR CENTRE OF GAVITY SHIFT.
C
C     INPUTS
C
C     AL2      ANGLE OF ATTACK ABOUT *M2* - POSITIVE WIND FROM LEFT
C     AL3      ANGLE OF ATTACK ABOUT *M3* - POSITIVE WIND FROM ABOVE
C     DL       ARRAY OF FOUR FIN DEFLECTIONS
C     MACH     MACH NUMBER (REAL)
C
C     OUTPUTS
C
C     C        ARRAY OF SIX AERODYNAMIC COEFFICIENTS
C
      REAL           DL(4)   , C(6)    , MACH
C
      COMMON/STABD/  A(6,5)
C
C-----------------------------COMPUTE EQUIVALENT CONTROL SURFACE DEFL-
C                             ECTIONS FROM THE FOUR SURFACE ANGLES
      DLA      = 0.25*(DL(3) + DL(4) - DL(1) - DL(2))
      DLY      = 0.50*(DL(1) + DL(3))
      DLZ      = 0.50*(DL(2) + DL(4))
C-----------------------------COMPUTE EACH MOMENT ASSUMING IT IS LINEAR
C                             IN EACH OF THE ARGUMENTS
      DO 110 J = 1, 6
      C(J)     = A(J,1)*DLA + A(J,2)*DLY + A(J,3)*DLZ + A(J,4)*AL2
     .                    + A(J,5)*AL3
  110 CONTINUE
      RETURN
C
      END

      FUNCTION DOT (A, B)
C-----------------------------COMPUTE VECTOR DOT PRODUCT OF TWO VECTORS.
C
C     PROGRAMMER V. B. WAYLAND
C
C     INPUTS
C
C     A AND B ARE ARRAYS OF LENGTH 3
C
C     OUTPUT
C
C     DOT IS A REAL FUNCTION RETURNING THE DOT PRODUCT OF A  AND B
C
      DIMENSION      A(3)    , B(3)
C
      DOT      = A(1) * B(1)  +  A(2) * B(2)  +  A(3) * B(3)
      RETURN
```

```
C
      END
      SUBROUTINE INV ROT(VIN, RMX, VOUT)
C-----------------------------INVERSE VECTOR ROTATION. INVERSELY ROTATE
C     AN INPUT VECTOR, VIN, FROM ONE COORDINATE SYSTEM THRU A TRANSPOSED
C     ROTATION MATRIX, RMX. THE NEW VECTOR IS VOUT.
C
C     B = (AB)T * A
C
C     INPUT
C
C     VIN        INPUT 3 VECTOR
C     RMX        3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT       OUTPUT 3 VECTOR
C
      DIMENSION      VIN(3)   , RMX(3,3), VOUT(3)
C
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(2,1)*VIN(2) + RMX(3,1)*VIN(3)
      VOUT(2) = RMX(1,2)*VIN(1) + RMX(2,2)*VIN(2) + RMX(3,2)*VIN(3)
      VOUT(3) = RMX(1,3)*VIN(1) + RMX(2,3)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
      SUBROUTINE  MMK(A, NA, B, NB, C, NC, RM)
C     1A-0     30 DEC 68       MAKE A DIRECTION COSINE MATRIX
C-----------------------------ROUTINE GENERATES A DIRECTION COSINE MATRX
C     BY ROTATING IN ORDER
C
C     1)ANGLE C ABOUT THE NC AXIS
C     2)ANGLE B ABOUT THE NB AXIS
C     3)ANGLE A ABOUT THE NA AXIS
C
C     INPUTS
C
C     ANGLES A, B, C IN RADIANS
C     NA, NB, NC - A NUMBER BETWEEN 1 AND 3 CORRESPONDING TO AXIS
C                  ABOUT WHICH EACH ANGLE IS ROTATED
C
C     OUTPUT
C
C     RM -- A 3X3 DIRECTION COSINE MATRIX
C
      REAL            AM(3,3) , BM(3,3) , CM(3,3) , RM(3,3)
      REAL            T(9)
C
C     NOTE FOR FORMING A DIRECTION COSINE MATRIX FROM EULER ANGLES THE
C     CONVENTION IS TO ROTATE ANGLE PHI   ABOUT THE NO. 1 AXIS, ANGLE
C     PSI ABOUT THE NO. 2 AXIS AND ANGLE THETA ABOUT THE NO. 3 AXIS
C
C-----------------------------GENERATE THE ROTATION MATRIX FOR EACH ANG.
      CALL ROTMX ( A, NA, AM)
      CALL ROTMX ( B, NB, BM)
      CALL ROTMX ( C, NC, CM)
C-----------------------------MATRIX MULTIPLY THE INTERMEDIATE MATRICES
      CALL  MML XY(BM,CM,T)
      CALL  MML XY(AM,T,RM)
      RETURN
C
      END
      SUBROUTINE  MML XY (X, Y, Z)
C-----------------------------MATRIX MULTIPLY ROUTINES FOR TWO 3X3
C     MATRICES. FIRST ENTRY CONTAINS NO TRANSPOSES
C
C     Z = (X) * (Y)
C
C     INPUT
C
C     X          FIRST 3X3 MATRIX
C     Y          SECOND 3X3 MATRIX
C
C     OUTPUT
C
C     Z          RESULTING 3X3 MATRIX WHERE
C     Z(I,J)  = X(I,1)*Y(1,J) + X(I,2)*Y(2,J) + X(I,3)*Y(3,J)
C
      DIMENSION      X(3,3)   ,  Y(3,3) , Z(3,3)
C
      Z(1,1)  = X(1,1)*Y(1,1) + X(1,2)*Y(2,1) + X(1,3)*Y(3,1)
      Z(2,1)  = X(2,1)*Y(1,1) + X(2,2)*Y(2,1) + X(2,3)*Y(3,1)
      Z(3,1)  = X(3,1)*Y(1,1) + X(3,2)*Y(2,1) + X(3,3)*Y(3,1)
      Z(1,2)  = X(1,1)*Y(1,2) + X(1,2)*Y(2,2) + X(1,3)*Y(3,2)
      Z(2,2)  = X(2,1)*Y(1,2) + X(2,2)*Y(2,2) + X(2,3)*Y(3,2)
      Z(3,2)  = X(3,1)*Y(1,2) + X(3,2)*Y(2,2) + X(3,3)*Y(3,2)
```

```
      Z(1,3)  = X(1,1)*Y(1,3) + X(1,2)*Y(2,3) + X(1,3)*Y(3,3)
      Z(2,3)  = X(2,1)*Y(1,3) + X(2,2)*Y(2,3) + X(2,3)*Y(3,3)
      Z(3,3)  = X(3,1)*Y(1,3) + X(3,2)*Y(2,3) + X(3,3)*Y(3,3)
      RETURN
C
      END
      SUBROUTINE  ROTMX( X, I, XM)
C     1A-0     30 DEC 68      ROTATION MATRIX
C----------------------------GENERATE BY STARTING WITH AN IDENTITY MATX
C     PUT THE COSINE OF ANGLE X ON THE DIAGONAL AND +SIN(X) AND -SIN(X)
C     ON OFF DIAGONALS
C
      REAL            XM(3,3)
      INTEGER          II T(3), III T(3)
C
      DATA     II T / 2 , 3  , 1  /
     .        ,III T/ 3 , 1  , 2  /
C
      SX      = SIN(X)
      CX      = COS(X)
      II      = II T(I)
      III     = III T(I)
C
      XM(I,I) = 1.0
      XM(I,II)= 0.0
      XM(II,I)= 0.0
      XM(I,III)= 0.0
      XM(III,I)= 0.0
C
      XM(II,II)= CX
      XM(III,III)= CX
      XM(II,III) =  SX
      XM(III,II) = -SX
C
      RETURN
C
      END
      SUBROUTINE  VEC ROT (VIN, RMX, VOUT)
C     1A-0   25 NOV 68          VECTOR ROTATION
C----------------------------ROTATE AN INPUT VECTOR, VIN, FROM ONE
C     COORDINATE SYSTEM THRU A ROTATION MATRIX, RMX.  THE NEW VECTOR IS
C     VOUT.
C
C     A = (AB) * B
C
C     INPUT
C
C     VIN      INPUT 3 VECTOR
C     RMX      3X3 ROTATION MATRIX
C
C     OUTPUT
C
C     VOUT     OUTPUT 3 VECTOR
C
      DIMENSION      VIN(3)   , RMX(3,3), VOUT(3)
      VOUT(1) = RMX(1,1)*VIN(1) + RMX(1,2)*VIN(2) + RMX(1,3)*VIN(3)
      VOUT(2) = RMX(2,1)*VIN(1) + RMX(2,2)*VIN(2) + RMX(2,3)*VIN(3)
      VOUT(3) = RMX(3,1)*VIN(1) + RMX(3,2)*VIN(2) + RMX(3,3)*VIN(3)
      RETURN
C
      END
```